Proceedings of the 9th Biennial ASME Conference on Engineering Systems Design and Analysis
ESDA08
July 7-9, 2008, Haifa, Israel

ESDA2008-59106

Towards a Unified Product and Project Lifecycle Model (PPLM) for Systems Engineering

Valeria Perelman **Amira Sharon** **Dov Dori**

Technion – Israel Institute of Technology

ABSTRACT

Developing and sustaining complex systems requires collaboration of multidisciplinary teams, coordination of processes, methods and tools, allocation of resources and utilization of adequate facilities within enterprises. The *system engineering management* comprises three intertwined domains: the *product***,** the *project* and the *enterprise*. Despite the obvious links between them, each is carried out using its distinct ontology and toolset. This conceptual separation hinders effective handling of the project and product lifecycle activities within the enterprise. Testing activities of complex products are focused on verifying the performance of increasingly large modules, from software and hardware components, through subassemblies to the entire operational system. *What* needs to be developed, tested, and delivered is determined by the product requirements, its functions, architecture, components, and their interactions. *When* each component should and can be developed and tested is determined by the project plan, which is dynamically re-estimated, re-evaluated, and re-planned depending on different parameters such as the project actual status compared with the plan, recourses availability, risks, technological breakthroughs or other impacting issues. *Whether* carrying out the development mission is feasible is determined by the responsible enterprise, its size, structure, management criteria, other projects running in parallel, commitments, and many other aspects. This paper introduces a unified project-product lifecycle management framework that attempts to address the problems cause by separating the product from the project that is supposed to deliver it within the executing enterprise.

1. INTRODUCTION

Systems engineering for developing and sustaining complex systems requires collaboration of multidisciplinary teams, coordination of processes, methods and tools, allocation of resources and utilization of adequate facilities within enterprises. *What* needs to be developed, tested, and delivered is determined by the **product** requirements, its architecture, and design. *When* each component should and can be developed and tested is stated in the **project** plan. *Whether* carrying out the development mission is feasible is determined by the responsible **enterprise**, its size, structure, management criteria, other projects running in parallel, commitments, and many other aspects. The *product***,** the *project* and the *enterprise* models used for planning and control of the efforts, undergo changes over time during each ones lifecycle.

It is widely acceptable that a conceptual modeling process that starts early on during the lifecycle has a great impact on the systems engineering success. The conceptual model describes the functionality of the system to be developed, defines system boundaries, its relation with the environment and its high level architecture. It comprises the basis for system level comprehension by different stakeholders, such as systems engineering managers, project managers, architects, and the multidisciplinary teams who carry out the project.

Conceptual modeling is performed at the early stages of the product's life cycle since the cost of problems detected during or after detailed design is prohibitively high, often requiring expensive backtracking to the earlier phases. Although a major goal of conceptual modeling is to serve during the early engineering phases, keeping conceptual model updated is essential, since it increases the system's reusability, provides a "big picture" system view during its evolution, and allows testing of newly introduced changes to the system at the conceptual level before investing in other activities related to their costly detailed design and implementation.

Efforts to create a set of processes to guide the system engineering management of the project and the product within the enterprise include the IEEE 1220 standard. This standard specifies the requirements for the systems engineering process (SEP) and its application throughout the product lifecycle. However, it does not attempt to define the *implementation* of each system lifecycle process, nor does it address its many cultural or quality variables. The ISO/IEC 15288 standard

defines a set of processes, termed system lifecycle processes, their outcomes, relationships and occurrence, but it does not offer a methodology.

The unified Product and Project Lifecycle Management (PPLM) framework aims to deliver both a methodology and an implementation environment, tested by a proof-of-concept, for managing complex enterprise-wide project-product systems. The resulting comprehensive unified model and supporting software infrastructure is expected to enhance product and project lifecycle management capabilities by providing a novel service-based framework for formal yet accessible design and maintenance of the combined product and project lifecycle. This paper introduces and elaborates on the PPLM framework. The vision of the PPLM is illustrated via a small scale example, followed by a discussion. Finally, future research directions are suggested.

1.1. Conceptual Modeling and Verification

Paradoxically, despite the need to reveal system problems as soon as they show up, current testing approaches, such as simulation, model verification, and run-time verification, do not operate on the conceptual model, but rather on the detailed design at a later stage or even at the implementation level. Moreover, most of the existing formal testing approaches and associated tools, such as *Modelica* and *Simulink*, require highly experienced professionals to carry them out and to analyze the results, making them too technical to be used during the conceptual modeling stage.

The need to verify a conceptual model can be addressed by simulated execution, iteratively updating the model and simulating it. This requires two basic components: a formal notation for conceptual modeling and a framework implementing the notation and providing model-based execution capabilities. Using these capabilities, the system's conceptual model can be executed by simulation, and its functionality and performance as well as its measurable characteristics can be. Continuous conceptual model updating through the whole system engineering lifecycle would maintain the system level view available for all the stakeholders throughout the system evolution and will enable conceptual model level simulation during its lifecycle. Both architectural and logical errors that are related to the conceptual model can be potentially detected before developing detailed design.

1.2. Product and Project Lifecycle Model (PPLM) principles

PPLM is aimed at fusing the *product* to be developed with the *project* including its milestones and deliverables as they are executed by the *enterprise*. The research aims to develop an underlying holistic conceptual model and supporting software for an integrated project and product lifecycle support environment. The PPLM research is expected to develop and evaluate an executable simulation method that operates at the conceptual level, based on a formal language which is appropriate for conceptual modeling. The characteristics required from the underlying language are simplicity and comprehensiveness, support of hierarchy and unambiguous definitions that enable execution.

Since creating the specially designed simulation model is usually expensive, efforts should be made to reduce the number of the retrials to construct the model. Starting with a lightweight simulation of the conceptual model potentially could reduce the cost of detailed simulation and the number of reconstructions.

2. BACKGROUND

In this chapter current executable modeling languages and simulation environments will be reviewed, as they are related to the research from an executable model-based framework perspective.

2.1. Evaluation Methods & Techniques

Various techniques are used in the systems engineering domain to perform combined qualitative and quantitative evaluations. Testing methods in the field of systems engineering are surveyed in [1]. Many of the methods are widely used also in the project management field. For example, risk management uses scenarios or questionnaires for risk identification, and business processes are validated using simulation. Questioning techniques, including questionnaires, checklists and scenarios, may be used to enhance discussion and comprehension of the system architecture or project plans. It is usually impractical to evaluate complex quality attributes using rigorous mathematical formulae and models. Collaborative efforts involve various system stakeholders who represent different viewpoints and raise potential system/project scenarios revealing system/project context-dependent risks and unsatisfied quality attributes. Previously defined questionnaires and checklists help define questions serving as a basis for further quantitative evaluations. They are not limited to system architecture, but also address processes used within their lifecycle. While a questionnaire is more general and more appropriate for the early phases in the product management lifecycle, checklists require more experience, thus they are more appropriate for the middle and final phases.

Observable measurements can be carried out using both general and domain-specific metrics, such as fan in/fan out of the components in the system architecture. Simulation and prototyping are other quantitative evaluation techniques. Creating a detailed prototype or simulation is often too expensive, but a detailed model increases system comprehensibility and enables more scientific analysis of the possible outcomes. Examples of simulation-based evaluation approaches include SystemC [2] for embedded systems performance analysis, while in the project management domain, Monte Carlo simulation [3] and PERT serve to estimate project completion time in the presence of uncertainty.

Benefits of early model-based validation and verification (V&V) include reducing risks related to the development of error-prone complex systems, saving both financial and time resources, working at the more abstract levels, filtering out

unimportant details, modeling with greater concern regarding the system requirements, identifying potential system evolution directions, prediction of possible changes to the design, and evaluating costs of making the changes.

Most of the currently used V&V techniques are applicable at lower levels of the system, such as hardware circuits [4], software coding [5], and formal verification at the algorithmic level [6]. Model checking tools face the state explosion problem, while theorem provers [7], [8] make the specification of the constrains to be proven complex, requiring special personal talents and high experience. Both checked constrains and system models must be specified precisely, making the process appropriate for hardware or code, but hardly executable at the system level.

MoDe [9] is a system-level design methodology that defines qualitative analysis using formal verification methods. However, it works on a limited set of system quality characteristics: responsiveness (timing fitting the requirements) and optimal partitioning (modularity).

2.2. Overview of Simulation Techniques and Tools

Since simulation approaches are widely used for both project planning and product architecture evaluations, a simulation module is highly likely to be a central building block in the PPLM framework. This section contains a short overview of the simulation objectives, a description of when simulation activities occur, and a short survey of leading simulation tools.

An analytical approach is usually based on a theoretical background and is therefore reliable, but it requires human experts. Moreover, the formalism of the required models renders it infeasible for complex systems such as PPLM-type systems. Simulation approaches, in contrast, do not require that the user constructs extensive mathematical models. While potentially less reliable, simulation is more flexible and applicable to a wide array of problems and systems.

Simulation of a system is usually built after constructing its conceptual and logical models. Simulation processes are used in the following three types of analysis: Descriptive analysis, enabling system validation and verification, prescriptive analysis, for system optimization purposes, and post-prescriptive analysis, also called "what-if" analysis, to predict system future modifications and their impacts.

Since constructing detailed simulations is too expensive and in most cases impractical, information is filtered out by a translation from the system conceptual level to the simulation model. General simulation model building blocks include the following system attributes, which define the system state at any point in time:

- events occurring at some time points, causing changes in the system state,
- entities or objects transferred through the system model,
- queues, where entities are waiting for some system state changes,
- creating – causing arrival of entities at some points of time,
- scheduling – assigning a new future event to an existing object, and
- system random variables specified by their values distribution.

There are two main types of simulation systems:

- Discrete event simulation systems, such as ProModel [10] and ARENA [11], which are based on a discrete and finite set of events, keeping a stable system state when no event occurs (e.g., the number of visitors in a bank changes only at the departure/arrival of client events),
- Continuous simulation systems, such as Stella and iThink [12], which enable continuous system state changes, such as the amount of water in the water reservoir

Simulators are also characterized as static (e.g., SimulAr [13] and Simulacón 4), simulators that disregard the time axis, such as the Monte Carlo simulation, vs. dynamic (e.g., VisSim [14]); stochastic simulators (e.g., @RISK and Crystal Ball [15]), where the output itself is "random", vs. deterministic simulators with no probabilistic components. There is also a variety of hybrid deterministic/stochastic simulators, such as Extend [16], which support both discrete and continuous systems, SEMoLa (Simple, Easy to use, Modelling Language) [17] for continuous/event driven, systems.

Other types of simulation systems include qualitative simulation (e.g., Qsim [18]), Web based and distributed simulation systems used in such fields as computer networking (e.g., Silk [19]), domain-specific simulators (for instance, Extend Industrial Systems library with linkage to MS Office), simulators supporting blocks with user-defined code in C/C++ (e.g., VisSim), Java, MATLAB, and others.

2.3. Model-based execution/simulation approaches: xUML and Workflow Engines

Recently, domain-specific simulators using abstract high level models, which are relevant to PPLM, have been developed in the fields of software engineering and business process IT engineering. This section briefly surveys these simulating approaches and tools.

The Model Driven Architecture (MDA) is a software design approach initiated by the Object Management Group (OMG) in 2001 [21]. MDA facilitates model-driven engineering of the object-oriented software systems. MDA distinguishes two types of the system models: platform-independent model (PIM), defining the system domain-specific but platform-independent model, and platform-specific models (PSMs), which is grounded to the platform-specific implementation. These models are specified using the UML notation.

Recently, the semantics of the UML has been extended by the Action Specification Language (ASL) [22]. A set of actions

defined by the ASL is based on a limited set of actions specified by the Precise UML group (PUML) [23], which worked on the discovery of UML semantic ambiguities. Currently, there is still no standard ASL, and different vendors have developed their own semantics.

With ASL, the UML semantics has been restricted in some ways. All these modifications were invented to enable UML model execution (xUML) [24]. The xUML specifies a deterministic system using a profile of the UML, the ASL language, and the Object Constrains Language (OCL). The xUML models are highly testable. They also support an MDA compilation process of the PIMs into the PSMs. There exist different tools and research prototypes supporting xUML simulation and verification, such as Bridgepoint [25] and Kabira Action Semantics [26].

Application of verification techniques in the context of MDA using the xUML models is explored in [27]. The verification process is based on the Temporal Logic of Actions (TLA), which was originally defined by L. Lamport in 1994 [28]. Another work based on the xUML models was implemented in the iUMLite [29] CASE tool. iUMLite supports model-based simulations using an xUML model of the system, initialization segments, including initial objects population, the initial status of the signal queues, the test methods, and user-defined scripts. iUMLite generates code based on the predefined initialization segments, test methods, and the xUML model. It supports running the code and includes a debugging mode with step-by-step progress that enables observation of the model objects and local variables.

The major criticism of xUML relates to its formalism and coding that is required of the system analysts and managers defining the MDA PIM models, who often do not have the appropriate programming skills. Moreover, the UML inherent multiple views make it difficult to understand clearly the various connections among the generated simulation code and the source xUML diagrams. Finally, the fact that there is no standard ASL semantics and the high software orientation of xUML make it difficult to base the PPLM unified model on xUML notations. However, partial integration with the xUML supporting tools may be one of the research future directions.

Business process modeling and simulations are widely explored approaches. Their aim is to bridge the differences among the various enterprise communities and stakeholders. The domain expert identifies the business needs, which are translated by the business analysts into the future business processes and system constrains. Further interpretations are made by the enterprise and software architects in order to construct the final solution architecture.

Business Processes Execution Language (BPEL) [30] is an XML-based language aiming to support programming in the large, i.e., describing the high level system state transitions and business processes. Business Process Modeling Notation (BPMN) is a graphical notation for business processes workflow representation developed by the BPMI [31] and maintained by the OMG. Both BPEL and BPMN address the problem of bridging the gap between business and technology people.

BPMN2BPEL is a tool for providing translation from BPMN to BPEL. However, due to lack of standardization, no strictly automatically generated synchronization can be achieved. BPEL4WS is an extension of BPEL for Web Services, which supports ways to define business process interaction protocols. WS-BPEL 2.0, which is based on BPEL4WS, has been accepted as a standard by the OASIS WS-BPEL technical committee in 2004.

The IBM WebSphere Process Server [32] is a complete BPEL engine running on top of the WebSphere Application Server Java EE platform. There are also many BPEL workflow engines, such as the ActiveBPEL Open Source Engine [33]. Its inputs include BPEL process definitions and WSDL files. The engine runs on any standard servlet container such as Apache Tomcat [34]. While executing, it takes care of persistence, queues, alarms, and many other execution parameters.

The most problematic issue related to workflow engines is their limitations with respect to human involvement as part of the simulation. For instance, BPEL does not define a standard work list service from which human tasks can be managed.

Both xUML simulation and verification tools and BPEL workflow engines incorporating web service invocations are possible candidates to serve as a basis for development of the PPLM executable framework. This might include both integration of the tools and adapting them to the PPLM needs. The lessons and techniques learned will help us determine to most suitable technological approach to constructing the optimal PPLM software environment.

3. CONCEPTUAL MODEL-BASED FRAMEWORK

The PPLM executable model-based software environment is designed to account for instances and be amenable for simulation as a basis for decision making. The conceptual model-based execution framework will serve as a foundation for a number of model-based activities during the systems engineering conceptual modelling process, including:

- Model-based requirements management
- Model-based test management
- Model-based risk management
- Model-based cost management (in the PPLM projects)

3.1. OPM Executable Framework

The candidate modeling notations to be the basis for the simulation framework were UML, xUML, SysML, Modelica and OPM. UML and xUML were not selected as they are software oriented and hence less appropriate to modeling of general systems. SysML was designed for general systems and its notation modifies a subset of UML diagrams, adding two new diagrams: Requirements Diagram and Parametric Diagram. Similarly to UML, the notation provides mainly aspect decomposition with hierarchical breakdown supported for some types of diagrams. Unfortunately, the multiple view model of SysML makes it difficult to comprehend the complete system,

which is one of the major tasks of the conceptual design. Moreover, unlike xUML, the notation does not use the formal activities specifications, and it allows free text for equation descriptions in the Parametric Diagram. This lack of formality makes it impossible to define SysML model based execution without significantly changing the notation. Finally, Modelica was dismissed as being too technical for wide user, and with too detailed design orientation and bottom-up approach, which is less appropriate during conceptual modeling.

OPM notation supports conceptual modeling for general systems. Its top-down approach includes refinement mechanisms of in-zooming and unfolding. OPM uses a single type of diagram to describe the functional, structural and behavioral aspects of the system. OPCAT is a CASE tool that implements the OPM approach. Its existing features and modules, such as an easy to use API, a basic animation module, and integration with files in the CSV format, reduce the research application development stage. CSV is commonly used by various project management tools, such as MS Project, and is useful for the PPLM case study construction. Hence, the proposed executable model-based framework and platform will be developed as an extension of Object Process Methodology (OPM) using the infrastructure provided by OPCAT. Nonetheless, we believe that the methods and algorithms that will be developed through the research will be reusable for development of conceptual model-based frameworks based on other modeling notations.

The execution/simulation module will extend OPCAT. The evaluating process, illustrated in Fig. 1, will operate on the OPM conceptual system model. The first subprocess of evaluating is optional and will enable importing predefined OPM constraints model during simulation/execution, supporting reusability of the testing functions by different system components.

An OPM-based query language will be developed as a part of the executable model-based framework. The language will enable focusing the simulation on the user-defined scenarios or processes interacting with filtered structural elements. All the activities will comprise information filtering subprocess.

The framework will enable instances-level simulation/execution. Since the populations of instances could be large, the process should treat the populations efficiently. For instance, uploading the instances from the DB as needed could address the problem.

After the user has defined the settings, the simulation/execution process can be invoked. The process should operate in both graphical and offline modes. While the graphical mode can be used as the conceptual model debugger, offline mode is used for simulation of the whole module numerically, collecting various system statistics and metrics.

The simulation/execution process will generate system traces. The traces will include an ordered set of the system states. Each state will be defined as the currently running processes, their duration, and the values of all the instances participating in the run.

Based on the collected traces run-time verification algorithms and statistics will be invoked. The process will generate a set of reports. Optionally, a translating process will then be invoked. Its output can be used as an input to a detailed simulation model that runs on external simulation tools outside the PPLM framework. Figure 1 is an OPM model of the PPLM evaluation in-zoomed.

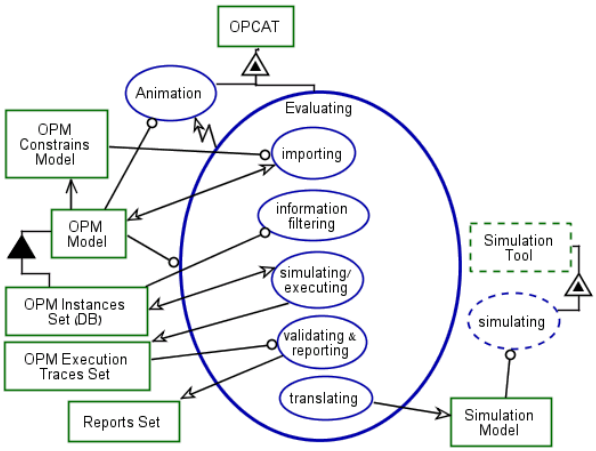

Figure 1: OPM conceptual model-based evaluating process in-zoomed

3.2. The PPLM Framework

The Project-Product unified managing process will be based on the Evaluating process described in the previous section. The process consists of the following sub-processes:

1) Semi-automatic matching of the various types of the PPLM relations among project and product artifacts supported by the project and product external tools.
2) Defining a library with system and project constraint models. This comprises of specifying functions that calculate values of parameters involved in constraints, such as the cumulative weight in a complex avionics hardware component constructed on sub-components, or the cumulative cost of a product by considering the costs of the individual parts and their integration. Another example is the risk of a project based on bottom-up calculations of risks involved with developing each one of the product's components and integrating them. A basic library of project and product constraints, including various project/product classifier attributes and related executable functions (estimators) will be configured as a reference to the PPLM model.
3) Specifying frequencies and probabilities for project events and the PPLM project processes duration, and optionally linking leaf processes of the project model with Web services.
4) Running a verification process on the PPLM product model and detecting unsatisfied requirements. The verification process could be continued by detailed product design

simulation. This will be achieved by exporting the PPLM model related to the product part into a simulation tool, such as Simulink or Modelica.

5) Running a PPLM evaluation process based on queries defined by the system engineering manager. The evaluation process will use execution capabilities of the framework. The simulation/execution will focus on the user-defined scenarios or filtered structural elements of the PPLM model during the PPLM evaluation process. Reports generated by the PPLM evaluation process will contain information regarding current state of the PPLM model and possible forecasting of the alterations.

Figure 2 summarizes the described refinement by the in-zooming of the PPLM managing process, which includes PPLM data linking (defining relations among various artifacts), specifying external library with the system constrains (setting PPLM constraints), validating the PPLM model using the defined constraints, optional translating of the model for the further detailed external simulating process, PPLM project semi-automatic evaluating using the expert-defined OPM queries, and defining the PPLM model subset that will be the focus of the evaluation (see Figure 2).

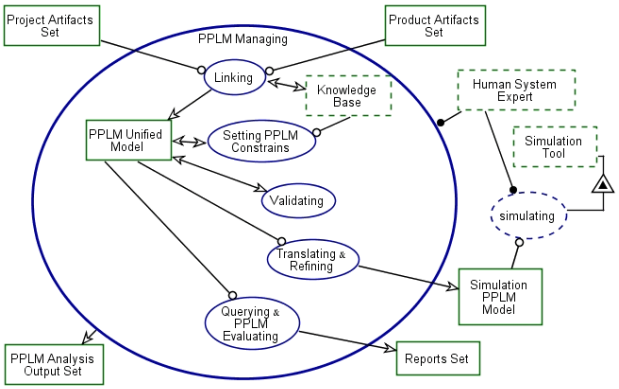

Figure 2: The PPLM managing process in-zoomed

4. PPLM EXAMPLES

4.1. Conceptual Design Review using the OPM Executable Framework

A hypothetical invocation of the OPM executable capabilities will be used to test a small scale system specified at the conceptual level. The subject system is described, and then the set of requirements used in the example are listed. Finally, the execution/simulation operating to test the requirements is described.

On-Line Ordering System (OOS)

OOS is a virtual Web-based store that serves as the mediator among its customers and suppliers. The system has no physical warehouse for the items it sells. The supplier receiving an order is responsible to send the physical package to the client. The OOS site visitors should be able to read the store guaranty, register themselves as suppliers for the selected items, and make their orders from the store.

For the sake of scalability and security, no constant data regarding the system clients is kept beyond the order processing. Supplier data is kept until the supplier removes himself from the system or has ignored the system appeals three subsequent times.

OOS Conceptual Model

The Product Ordering and Supplying Managing processes are described as part of the OOS conceptual model in Figure 3 and Figure 4. The Product Ordering process starts with the Ordering subprocess invoked by the Client invocation. The Client then selects items to be purchased from the OOS Catalog, inserts his maximal price for each one of the selected items, and fills in the client form, including Credit Card information. The Ordering process finishes by either initiated Order in the "in order" state and a digital Invoice whose printout is sent to the physical Client, or by Reject Notification if any problem with Client details or with the Order were detected.

The Supplying Managing process is invoked subsequently to the Ordering process according to the OPM in-zooming rules. This process tries to select the appropriate supplier who could supply the whole Order within Client's sum of the maximal prices for the Order Items at most three times. Each Supplier selection is followed by the Supplying Managing process by the subprocess Ordering from Supplier. The sub-process waits at most two weeks for the supplier reply. If no reply from him was received, the OOS manager is alerted and asked if the next supplier should be tried.

The Supplying Managing process finishes with Order Canceling or Supply Approval. In the first case, the Client receives system error notification, whereas in the second case the physical Order is sent to the Client.

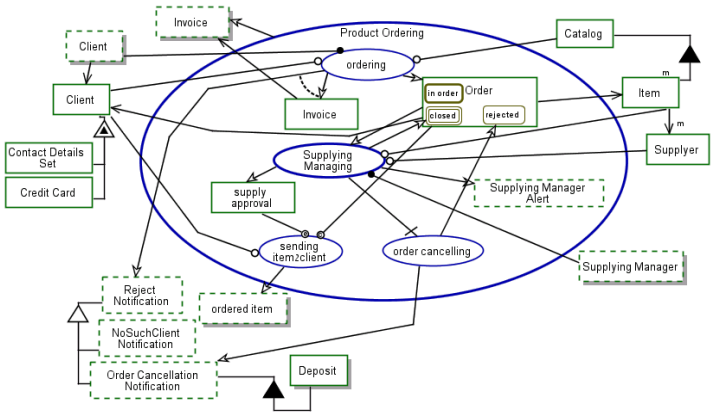

Figure 3: Product Ordering in-zoomed

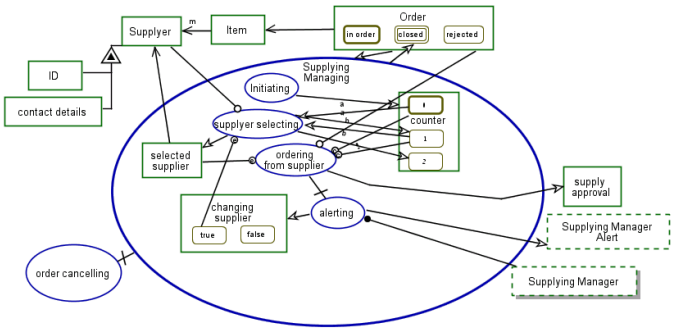

Figure 4: Supplying Managing Process in-zoomed

OOS Requirements Testing

Few functional and non-functional requirements were selected to be tested with the PPLM framework. The list of requirements (constraints) is followed by an explanation on how they will be tested using the framework.

1. **Functional Path Constraint:** In case of failure during the Ordering registration process, which is the initial sub-process in Product Ordering, if Reject Notification was generated, no Order instance and Invoice instance should be created and the Product Ordering process instance must be stopped.
2. **Non-Functional Process Constraint:**
    a. During selecting of an appropriate supplier, the same supplier won't be selected twice in the context of the same Supplying Managing process run (instance).
    b. The selected Supplier must be able to supply all the items in Order.
    c. The Order price shouldn't be higher than the sum of the maximal prices for each item in the Order selected by the customer.
3. **Data Requirements**:
    a. Maximal price for each item in the client order should be kept during the product ordering
    b. Data regarding suppliers and the supplied by them items and their prices must be kept in the system
4. **Technical Requirements:** While many Product Ordering process instances may run simultaneously, the same Supplier shouldn't be selected more than ten times at each run. This constraint is needed to eliminate cases when the supplier mailing server decides that the OOS system attacks the server.

Requirement #1 could be checked using the simulation's visual mode. To achieve a more effective and quick debugging, the user specifies in the new framework a graphical zooming level and selects a process that must be passed by all the simulated run traces. The agent link from the external Client to the Ordering process should be assigned with the frequency value and maximal number of invocations through the simulation.

Since ordering by default has no process body, the next step in the control flow is selected uniformly. Thus, in few runs users could note that the requirement was unsatisfied. According to the model, the Ordering process generates Reject Notification or Invoice, and Order. Since the sequential process Supplying Managing requires only the Order object, it will take place after all the Ordering process invocations. To eliminate this problem, OPM paths labeling technique can be used. However, for more complex logical formulas even this technique must be combined with pseudo-processes and in-zooming mechanisms. Coding inside the ordering process body can be more efficient in such cases. The proposed Ordering process body will be:

```
Input: Failure
If (Failure && In [Product Ordering]){
        new Reject Notification;
        halt [Product Ordering];
}else{
        new Order;
}
```

The constraint described in the requirement can be formally described as the sequence of run-trace events and checked using run-time verification methods.

Suppose that the simulation events include: (1) the process is invoked or stopped (unary operators), (2) the process creates/destroys an object (binary operators with right and left parameters). Suppose also that any process/object instance has its unique ID. Then the constraint could be formulated in the following manner:

(1) *Foreach trace t: [[Ordering process] Creates [Reject Note object]] must be followed by Stopped [Product Ordering process]* |
Where [Product Ordering instance ID] = [Ordering Father instance ID].

Requirement #2 could not be resolved without running the Supplying Managing process on an instances population. For each run, the states of all the instances in focus for every step will be saved and analyzed offline.

Requirement #3 is not satisfied by the presented model. However, these are examples of requirements that can be tested statically without using the simulation capabilities.

Requirement #4 is an example of system-level rather than a single path constraint of checking the number of selections for each supplier in each step. This requires more extensive run of the product ordering process concurrently. Mathematical formula or code will be specified in the supplier selecting process to define the selecting policy. Simulation with a set of supplier instances of the process will be invoked. For each supplier instance the user could check his resources usage online during the run till each point.

4.2. A PPLM Project Example

The PPLM example includes the hypothetically invoked evaluation methods, the PPLM product description, the product requirements and its partial conceptual model, and the testing process based on the OPM executable capabilities.

The evaluation methods

The following methods will be used to evaluate the PPLM environment:

1) Product validation using the executable capabilities of the PPLM framework carried out at the conceptual top level architecture of the model.
2) Forecasting the project progress and meeting milestone dates by analysis of the currently non-supported requirements.

Case study description

The project aims to develop a modern braking system for automatic vehicles. The system consists of an antilock braking system (ABS), a traction control system (TCS) and brake assisting system called BAPlus with an advanced feature for approximation of the distance to other vehicle involving a radar component.

The required braking system constraints

Preventing wheel locking, Traction control – preventing loss of traction, and Break assisting – detecting panic brake or emergency situation, in which another vehicle is too close and automatic assistance in reduction of the stopping distance are the primary braking system functions.

The system quality requirement:

R 1 The whole system weight shall not be greater than 2 kilograms.

R 1.1 Derived Requirement: ABS System shall weigh at most 1.5 kilograms.

The business requirements:

R 2 Cost reduction

R 2.1 The derived system technical requirements: The brake actuator and the wheel sensor system (WSS) shall be shared by the ABS, TCS, and BAPlus sub-systems.

R 2.1.1 ABS and TCS shall be interfaced.

R 2.1.2 ABS and BAPlus shall be interfaced.

R 2.2 The total cost of the braking system shall not exceed $800,000.

R 2.2.1 ABS sub-system cost shall not exceed $500,000.

The technical constraint:

R3 TCS and BAPlus shall have an identical type of connectors that will interface with the WSS components of the ABS sub-system. Note: we suppose that the interfaces and the system architecture conceptual modeling phases were carried out before the exemplified evaluating process invocation. Hence the current constrain was defined beforehand.

The Braking System Architecture

The OPM System Diagram—the top level OPD in Figure 5 depicts the architecture of the system. Following OPM guidelines for top-down functional decomposition, the Brake Controlling process is the central process of the system, referring to the main function of the system. The Braking System consists of three subsystems: ABS, TCS and BAPlus. The Braking System is the instrument for the Brake Controlling process and is part of the whole Car system, which in addition consists of four wheels and is characterized by the Velocity attribute. Both states of the velocity and of the wheels can be influenced by the Braking System Controlling process. The process can be invoked externally by the Operator or internally by the Emergency Braking Event, generated by the BAPlus subsystem. The event is generated as a result of detecting by the BAPlus radar that the vehicle is too close to another object.

OPD describing the ABS top level architecture is shown in Fig. 6. The subsystem implements the required system function F1 – preventing wheel locking. Its central process is the ABS Antilocking function, which relates to the F1 function. Similarly, F2 and F3 are the functions related to the central processes of the TCS and BAPlus sub-systems, respectively.

ABS is composed of ECU and a set of four Hydraulic Modulators and four WSS components. Each one of the modulators actuates exactly one wheel of the Car; similarly each one of the WSS components monitors exactly one wheel. ECU is composed of the Microprocessor and ABS software running on the microprocessor and invoking the ABS set of rules in accordance with the difference in the wheel speeds detected by the WSS components. The rules generate commands executed by the Hydraulic Modulators. For instance, the Hydraulic Modulator of some Wheel may assist in breaking that Wheel if its speed was essentially higher than that of the other wheels. The ABS Antilocking process, which controls the car wheel speeds, is an operation (process feature) of the ABS subsystem. It is instrumented by the WSSs, Hydraulic Modulators and the Microprocessor.

The additional attributes of the ABS subsystem are its cost and weight. We assume that all the parts (physical as well as informatical, or software components) are characterized by the cost attribute and its physical parts are also characterized by their weights.

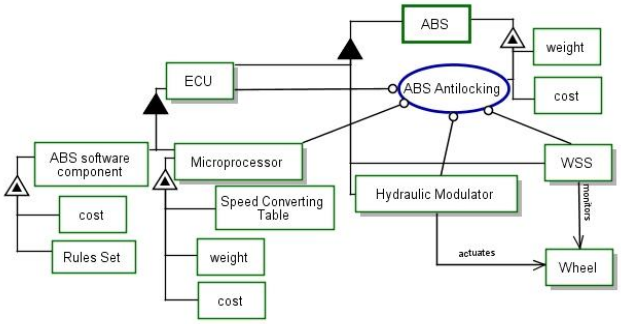

Figure 5: ABS unfolded

PPLM Braking System Parameter Equations Library

In the OPD depicted in Figure 7, few of the equations addressing technical constrains are modeled. These constrains relate to the requirements R1, R1.1 and R2.2. The model specifies calculating the weight and cost attributes of the system, subsystems and components.

For simplicity, we assume that the weight and cost of each system, subsystem, and component can be calculated as the sum of the weights or costs of their lower level parts plus an integration weight and an integration cost, whose values are predefined.

The OPM executable notation will include well-defined basic executable processes for the OPM basic types (e.g., integer, float), such as Adding, which is the plus arithmetic operator, from which all the other calculations will be hierarchically composed. For example, the averaging process operating on a set of integers will include adding them and dividing by the size of the set. In this example, we assume that the weight and cost attributes are of the double (in the Java annotation) basic type, and that the Weight Calculating and Cost Calculating processes use the Adding process.

The executable capability of the framework above makes it possible to execute the model. The executable model of this particular system will be embedded in the generic PPLM model and executed there as part of the large integrated project and product model. This will enable tradeoff analysis of various constraints. For example, if the resulting system is too heavy by 100 Kg, redesigning components for lighter weight will require 4 weeks in the project's critical path. Is the delay justified? Perhaps continue with the heavy parts now and in parallel redesign for lighter parts?

Figure 8 shows the way to import from the current library and invoke the Weight Calculating and Cost Calculating external processes, and checking the appropriate constrains. The match of the elements is done using the meta-tags used in the original system architecture model.

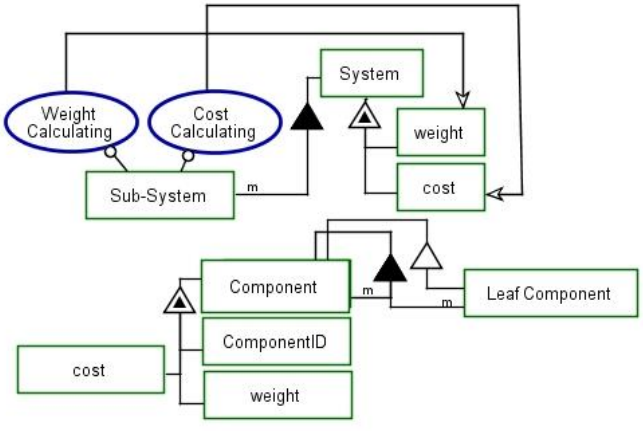

Figure 6: Weight and Cost Calculating functions: An example of a PPLM Parameters Library

Evaluation of quantitative non-functional requirements exemplified

The OPD depicted in the Fig. 8 belongs to the PPLM model of the braking system. This model represents the ABS subsystem and its relation to one of the technical constrains. The technical constrain, represented by the OPM object R2.2.1, stands for the requirement R2.2.1—ABS sub-system cost shall not exceed $500,000. It is modeled with the corresponding validating process, involving the Cost Calculating process from the PPLM Parameter Calculations Library exemplified in Fig. 7.

The assumption is that the braking system architecture is tagged using the PPLM ontology and the PPLM Parameter Calculations Library was defined using the same ontology. Thus, the matching of objects involved in the system process is done in correspondence with their PPLM roles as defined in the PPLM Parameter Calculations Library.

Execution of the validating process using the future OPM executable capabilities and including instances population with predefined start values could generate the actual cost of the ABS subsystem and determine whether requirement R2.2.1 is satisfied or violated by this subsystem. This type of the product model evaluation could be used by the system engineer and the technical management to comprehend the current state of the product and its implications at different phases of the project. This could be used in constructing prognoses of the further project evolution, as described in the following section.

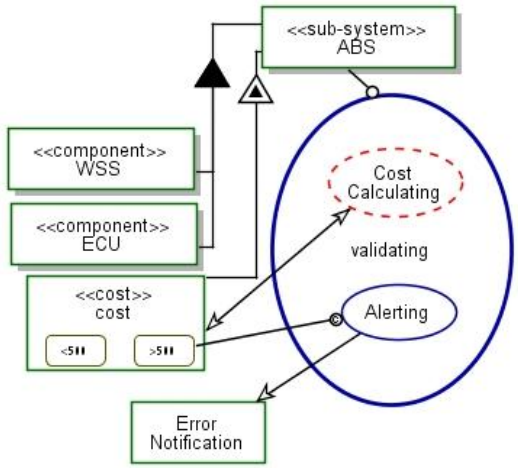

Figure 7: Validating the R2.1.1 requirement in the PPLM model using the externally defined cumulative functions

Project Prognosis

In the following evaluation method example we assume that data depicted in the OPD of Figure 9 was constructed semi-automatically as a subset of the unified PPLM model for the braking system example. Since the example illustrates construction of the project prognosis, the time scale is essential. The assumption now is that the system engineers and project managers are currently executing ABS & TCS integrating task. They have used the PPLM model and invoked validating

processes that had been defined for all the technical requirements related to the ABS and TCS subsystems. Based on the results of the processes and the original project plan and using the PPLM analyzing module, they find possible future influences of all the requirements that have not been satisfied by now.

Suppose requirement R2.1.1 – ABS and TCS shall be interfaced – was unsatisfied since the integration of the two subsystems TCS and ABS failed due to some architecture-level inconsistencies. This failure was detected through invocation of the OPM execution, simulation or animation modules on the unified PPLM model. The technical constraint R3 defines a strong relation between R2.1.1 and R.2.1.2 – ABS and BAPlus shall be interfaced. For the sake of simplicity, we assume that failure to integrate ABS with TCS means that the ABS interface was wrongly specified; thus, failure to satisfy the requirement R2.1.1 means failure to satisfy the equivalent requirement of the ABS integration with the BAPlus sub-system.

The PPLM model keeps information regarding all the complex dependencies: Relation between R2.1.1 and R2.1.2, relation between R2.1.1 and the implementing ABS with TCS integration task, between R2.1.2 and ABS with BAPlus integration task, etc. Using the above information, managers are able to plan the improvement actions, such as inserting new task to the project. Assume a new task named Fixing the ABS Portability was proposed. Execution/simulation of the updated relevant subset of the PPLM model will reveal the impossibility of arriving at the next milestone on time if the fixing task duration will be longer than X days. Thus, due to the well-defined relations between requirements and system components, between requirements and tasks, PPLM framework could forecast possible failure to meet the next milestone, ABS & BAPlus review task. The calculated starting date of the task could be compared with the originally planned one. Note that without knowing the relations specified by the unified PPLM model, such forecasting should be carried out manually and be less accurate, especially for large scale systems.

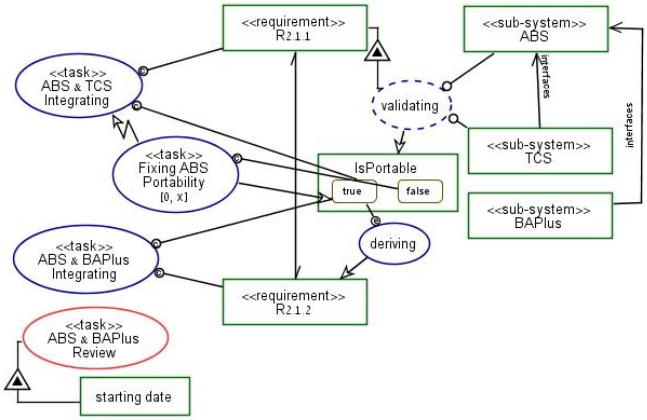

Figure 8: Partial PPLM model of the Braking System

5. SUMMARY

The PPLM research aims to define, implement and assess a methodology and a framework for joint project and product lifecycle management (PPLM). PPLM is expected to increase the likelihood of success of both the product under development and the project within which the product is developed within an enterprise. The research will contribute to bridging the gaps between the project and the product lifecycles by developing and evaluating a systematic approach—the PPLM methodology. The PPLM framework will integrate project and product technical and managerial data, information, and knowledge within a coherent unified model. As one of its most important features, the framework to be developed as part of this research will support model-based execution and simulation capabilities that rely on detailed data at the instance level. This will enable fine-grained analysis based on actual data, providing for "what-if" simulations.

Based on PPLM ontology, this research will focus on developing and assessing the conceptual modeling theory through the software infrastructure. The research will solve the theoretical problems associated with PPLM execution, and deliver the software framework required for simulating, monitoring, and analyzing the PPLM methodology.

The infrastructure will provide a solid basis for the next level of complexity, which is beyond the scope of this research—the Enterprise Project and Product Lifecycle Model (EPPLM). EPPLM will extend the PPLM model, which deals with a single project, to handling multiple projects within an organization, where these projects, each possibly at a different lifecycle phase, compete for enterprise resources. While more complex, the EPPLM tackles a more realistic problem and hence is of potentially higher interest for the enterprise.

REFERENCES

[1] Abowd, G. et al. *Recommended Best Industrial Practice for Software Architecture Evaluation* (CMU/SEI-96-TR-025). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1996.

[2] Grotker, T., Liao, S., Martin, G., Swan, S. System Design with SystemC. Kluwer, Dordrecht (2002).

[3] Pllana, S. History of Monte Carlo method. http://www.geocities.com/CollegePark/Quad/2435/index.html, August 2000.

[4] Hu, A. J. Formal Hardware Verification with BDDs: An Introduction, IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing (PACRIM), pp.677-682, 1997.

[5] Hoare, C. A. R. An axiomatic basis for computer programming, *Communications of the ACM*, 12(10):576–585, October 1969.

[6] Rushby, J. M. and von Henke, F. Formal Verification of Algorithms for Critical Systems, IEEE Transactions on Software Engineering, v. 19 n.1, p. 13-23, January 1993.

[7] Lusk, E.L. and Overbeek, R.A. An LMA-Based Theorem Prover. Technical Report ANL-82-84, Argonne Nat'l Laboratory, Dec. 1984.
[8] McCharen, J. D., Overbeek, R.A. and Wos, L., "Problems and Experiments for and with Automated Theorem-Proving Programs", IEEE Transactions on Computers, Vol. C-25, pp. 773-782, August 1976.
[9] Romberg, J. Slotosch, O. and Hahn, G. *MoDe: A Method for System-Level Architecture Evaluation.*
[10] ProModel - http://www.promodel.com/products/promodel/
[11] Arena – homepage: http://www.arenasimulation.com/
[12] Stella & iThink – homepage: http://www.iseesystems.com/
[13] SimulAr – homepage: http://www.simularsoft.com.ar/
[14] VisSim – homepage: http://www.vissim.com/
[15] Crystall Ball – homepage: http://www.crystalball.com/crystal_ball/index.html
[16] Extend – homepage: http://www.imaginethatinc.com/
[17] SEMoLa – homepage: http://www.dpvta.uniud.it/~Danuso/docs/Semola/homep.htm
[18] Qsim – homepage: http://www.cs.utexas.edu/users/qr/QR-software.html
[19] Hearly, K. And Kilgore, R. Silk: A Java-based process simulation language. In *Proceedings of the 1997 Winter Simulation Conference*, ed. S. Andradottir, K. Healy, D. Withers, and B.L. Nelson, 475-482. Institute of Electrical and Electronics Engineering, Piscataway, New Jersey.
[20] Simulink – homepage: http://www.mathworks.com/products/simulink
[21] Model Driven Architecture – A Technical Perspective, OMG AB paper, 2001.
[22] Wilkie, I. King, A. Clarke, M. Weaver, C. and Rastrick, C. UML ASL Reference Guide, Kennedy Carter. 2001.
[23] pUML – homepage: http://www.puml.org
[24] xUML –Mellor, S. J. and Balcer, Mark J. Executable UML, Addison-Wesley, 2002.
[25] Project Technology, Inc. BridgePoint Action Language (AL). http://www.projtech.com
[26] Kabira Technologies, Inc. Kabira Action Semantics. http://www.kabira.com
[27] Graw, G. and Herrmann, P. Verification of xUML Specifications in the context of MDA. In: *Workshop in Software Model Engineering (WISME@UML'2002),* Dresden, October 2002.
[28] Lamport, L. The temporal Logic of Actions, ACM Transactions on Programming Languages and Systems, vol. 16, no. 3, pp. 872-923, 1994.
[29] iUMLite – homepage: http://www.kc.com/ , 2007.
[30] Business Process Execution Language for Web Services version 1.1 IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. Available http://www.128.ibm.com/developerworks/library/specifications/ws-bpel/ , 2006.
[31] BPMI – homepage: www.bpmi.org/ , 2007.
[32] IBM WebSphere – homepage: www.ibm.com/websphere , 2007.
[33] ActiveBPEL – homepage: http://www.active-endpoints.com/active-bpel-engine-overview.htm , 2007.
[34] Apache Tomcat – homepage: http://tomcat.apache.org/, 2007.