Propean, a RT-UML based Approach to Help Manager's Decision-making

Antonia Bertolino
Istituto di Scienza eTecnologie dell'Informazione
ISTI-CNR, Pisa, Italy
+39 050 3152914
bertolino@iei.pi.cnr.it

Eda Marchetti
Istituto di Scienza eTecnologie dell'Informazione
ISTI-CNR, Pisa, Italy
+39 050 3153467
e.marchetti@iei.pi.cnr.it

Raffaela Mirandola
Dip. Informatica, S& P
Università di Roma TorVergata
Roma, Italy
+39 06 72597381
mirandola@info.uniroma2.it

ABSTRACT

We pursue the usage of classical performance engineering to aid the project manager in making sound, reliable schedule predictions, and in optimizing personnel utilization during software development. We assimilate the project teams to the processing elements of a performance model, and their activities to the tasks to be accomplished within established time intervals. We then show how by means of basic performance analyses, different workflow assumptions can be explored and their consequent outcomes automatically derived: by looking at the analysis results, the manager can thus take an informed, more responsible decision. The proposed approach is called Propean (for Project Performance Analysis). To use Propean, the manager does not need to be an expert of performance modeling notations as we provide a front-end interface based on a subset of Real-Time UML, the OMG standard profile specialized for schedulability, performance and timeliness. To illustrate Propean application, in this paper we model the case of a manager that must decide a release date for a product undergoing the testing phase.

Keywords

Product Release, Project Management, Real-Time UML, Software Performance Engineering.

1. INTRODUCTION

Performance Analysis refers to the vast body of techniques and procedures used to assess and predict the time dependent behavior of computer devices and networks. The incorporation of such techniques within the development process is referred to as Performance Engineering. The area has been the subject of extensive research since 1970s and is today a quite mature discipline, with a variety of modeling approaches and of solving techniques available [15][19].

However, it is only recently that performance engineering has started gathering interest also in the design of software systems, to model and estimate their quality of service (QoS) requirements. Perhaps the most compelling reason why performance analysis techniques have not yet become a key tool for software engineers as they ought to be is the extremely specialized formalisms required [22], such as queueing networks, stochastic Petri nets and Markov models.

In fact, a great effort is currently undertaken within the software engineering community for providing performance techniques with front-end representations that are easier to use, and closer to the commonly employed design notations. The most prominent example in this regard is the Unified Modeling Language (UML), which is the emerging standard notation in industry for system analysis, design and implementation [25], [37]. On one side several authors have proposed ad hoc methods for performance modeling of systems by means of UML diagrams (see papers in [34], [35], and [10]); on the other hand the Object Management Group (OMG) has recently adopted as a standard the RT-UML specialized profile [36] with modeling extensions that can handle timing specifications of services, mechanisms and resources (logical and physical); concurrency and scheduling; software and hardware infrastructures and their mutual mappings. Besides it allows for the flexible introduction of more specialized notations where necessary.

This paper embraces the trend of using UML as an input modeling notation towards performance models, but proposes an unusual application of performance engineering techniques: i.e., as an aid to project managers for decision making regarding the organization of teams and tasks within the software development process.

Project managers are faced with difficult decisions all along software development. In particular, they are in charge to judge whether the resources assigned to a specified task are adequate or whether under the existing organizational schemes the predicted time schedules will be met. Making such decisions is very difficult, because the involved processes are highly complex: the influencing factors (both human and technical in kind) are many, and in most cases not easily measurable or predictable.

We believe that, bypassing the classical application domain of computers operating behavior, performance analysis techniques could play an important role in such assessments. To achieve this, we assimilate the project teams to the processing elements of performance models, and the development activities to the tasks to be performed by those processing elements within established time intervals.

Following this metaphor, we show that well known techniques from performance analysis can be usefully and quite naturally adapted to tasks of relevance for software managers, such as assessing the time to completion of specified activities, handling personnel multitasking over different projects, optimizing the workloads in development cycles, deciding about products release, and similar issues.

The advantages of using performance analysis techniques in project management are many. We highlight the intrinsic capability to handle multiple parallel projects and their mutual interfering in schedule and resources usage. Moreover, the obtained predictions rely on a solid mathematical background and have a statistical validity.

A caution word is needed: performance techniques can provide the necessary analytical support to substantiate decisions, but the role of manager's expertise remain essential for tuning the input models with the proper parameters values, and for comprising the human and the unpredictable aspects. In brief, we propose performance analysis as an *aid*, and *not as a substitute*, for expert managers.

Precisely, we aim at building a setting in which managers develop a model of the flow of activities and of tasks distribution among personnel, using familiar notations and tools, and then a tool automatically translates the model in a format that is processable by standard performance analysis algorithms. Our approach is called Propean (for Project Performance Analysis).

The idea of using performance techniques in project management actually is not completely new, but applications so far have been limited to the case of a single project at a time, as for example in [1], or have been developed to handle specific situations, as in [2] for simulating the performance of cooperating maintenance service centers geographically distributed, and not as a general approach. In contrast, Propean provides a generic solution that can handle multiple projects and can be applied to any situation and workflow of activities. Moreover, to the best of our knowledge, our work is the first attempt to develop an approach based in the standard RT-UML profile.

More precisely, the transformation from the input model relies on an earlier existing method [10] that used the standard UML augmented with ad hoc annotations. In Propean, we have adapted that method to RT-UML.

It is a goal of this research project to carry on several trial applications of Propean and develop the relative RT-UML input models, so that blueprints for various plausible contexts in project management are already on hand, as aside documentation of the approach. In this perspective, we have developed a

reference model in [3] concerning a waterfall development process; another scenario is proposed here[1] to support the decision of releasing a product based on the analysis of trouble reports, and a more general case study encompassing the modeling of the whole Rational Unified Process (RUP) is currently under study [5].

1.1 Paper structure

Propean uses the Real-Time UML profile for the performance modeling of project management contexts. RT-UML is introduced in the next section with special emphasis to the Performance Analysis profile and the introduced annotation.

As an illustrative example, we consider a case study of a manager that must decide whether a product is ready to be released based on the status of the trouble reports compiled during the test and debug phase. The case study is described in Section 3.

In Section 4 we outline the steps of the proposed Propean methodology. Section 5 presents the application of Propean to the case study and Section 6 illustrates an analysis of the obtained results. The architecture of a tool realizing Propean is outlined in Section 7. Related work is overviewed in Section 8, while Section 9 presents conclusions and future work.

2. FROM UML TO RT-UML

The UML modeling language is rapidly becoming the standard notation for analysis, design and implementation of Object-Oriented systems. We assume in this paper that the reader is familiar with the UML notation, and recall in this section only the main characteristics of those UML diagrams (namely, Sequence Diagram and Deployment Diagram) directly involved in the Propean methodology (for more information see the UML User Guide (1999) [25] and UML 1.4 Documentation [37]). A Sequence Diagram *(*SD) shows an interaction between system elements, arranged in a time sequence. In particular, it shows the class instances (objects) participating in the interaction along their "lifelines" and the stimuli (messages) they exchange arranged in time sequence. The SDs do not show the associations among the objects, but can provide specific information about the order in which events occur.

[1] A preliminary version of this paper can be found in [4]

A Deployment Diagram (DD) shows the configuration of run-time processing elements and the software components, processes and objects that live on them. It is a graph whose nodes are connected by communication associations. Nodes may contain component instances (indicating that the component lives or runs on the node), and component instances, in turn, may contain objects (indicating that the object is part of the component). The DD can therefore show the mapping of components to processing nodes.

Although UML is generally recognized as a useful tool for modeling the functional characteristics of a system (e.g., see papers in [31], [32] and [33]), historically it had ignored non-functional requirements, such as response time, availability, throughput and bandwidth. These constitute important system features, nowadays often referred to in abstract as the QoS (Quality of Service) characteristics.

By general consensus the UML lack of a quantifiable notion of time and resources was felt as "an impediment to its broader use in the real-time and embedded domain" [26]. As reported in [26], in 1999 to cope with the needs from this key area, the Analysis and Design Platform Task Force of the OMG issued an explicit request for proposals[2] (RFP) for a UML domain-specific interpretation (to be fully conformant with the UML standard) capable to deal with non-functional requirements.

2.1 RT-UML

In response to the OMG RFP, a working consortium of OMG member companies proposed a UML Profile for Schedulability, Performance and Time (RT-UML), which has been recently adopted as an OMG standard profile [36].

Presenting a detailed overview of the RT-UML profile is out of the scope of this paper; we give here only the essential background needed to understand the RT-UML features we use in the Propean methodology. For major details we refer to [36].

RT-UML is not an extension to the UML metamodel, but a set of domain profiles for UML allowing for the construction of models that can be used to make (early in the life cycle) quantitative predictions regarding the characteristics of timeliness, schedulability, and performance. In particular, effort has been spent both to enable predictive quantitative analyses (e.g., the ability to determine the schedulability of a

[2] OMG document number: ad/99-03-13

planned piece of software or its response time), and to model QoS aspects, such as deadlines and priorities.

The idea underlying the RT-UML is to import, as annotations in the UML models, the characteristics relative to the target domain viewpoint (performance, real-time, schedulability, concurrency), in such a way that the various (existing and future) analysis techniques can usefully exploit the provided features. Generally domain viewpoints are not often used in practice because they require high expertise and specialized knowledge. The intent of RT-UML profile is to overcome this problem by providing a single unifying framework to encompass the existing analysis methods, still leaving enough flexibility for different specializations. At the core of the profile is the *general resource modeling* framework, which provides a common model of resources and of their QoS attributes. Then, based on this common framework, more specific sub-profiles are defined, i.e., "profile packages dedicated to specific aspects and analysis techniques". Their purpose is to specialize the generic concepts to better represent the needs of a specific domain, i.e., to derive a *conceptual domain model.*

The general resource modeling framework itself consists of three sub-profiles dealing respectively with resource modeling, concurrency and time-specific concepts. In the next section, we focus in particular on the RT-UML sub-profile we refer to for Propean, i.e., the performance analysis (PA) profile.

2.2 Performance Analysis profile

The PA profile is specifically designed for capturing the performance requirements and specifying the QoS characteristics or execution parameters. At a high abstraction level, the concepts characterizing the PA profile are:

- The *scenarios*, i.e., ordered sequences of steps, describing various dynamic situations involving the usage of a specified set of both processing and passive *resources* under specified *workloads* (i.e., the load intensity and the required or estimated response times for the scenario). In particular we can distinguish between: a *closed workload*, in which a fixed number of requests cycles while the scenario is executed, and an *open workload*, in which the requests arrive at a given (predetermined) rate.

  A step in a scenario is characterized by its mean execution number (i.e., the mean number of times it is repeated when executed) and the host execution demand (i.e., the execution time taken on its host devices) and might involve multiple concurrent threads, due to forking.

- The *resources,* i.e., the servers in a performance model that can be active or passive. The active resources are usual servers characterized by the *service time,* i.e., the execution demand of the steps that are hosted by resources, while the passive resources can be acquired and released during scenarios and are characterized by the holding times.
- The *performance measures* of the system that include: resource utilizations, waiting times, execution demands, and response times. Each of these values can be: derived from the system requirements or performance constraints (e.g., response time for a scenario); estimated on the basis of experience or previous knowledge (e.g., execution demand); directly measured or simulated.

The RT-UML PA profile provides UML extensions to deal with the above notions of scenarios, resources, and workloads and the associated attributes (in the following, PA attributes), so to allow for extensive and wide-ranging performance analyses. In our methodology, we are actually interested only on a small subset of these extensions.

PA scenarios can be modeled following either a Collaboration-based approach or an Activity-based approach (as in [21]). In the tradition of [10], we take here the former approach, and represent a scenario by an annotated Sequence Diagram. The usage of Activity graphs might present some advantages in expressiveness [36] (modification of the Propean approach to allow usage of Activity graphs is part of our future plans).

2.3 Annotations used

We report below a short description of the subset of PA annotations we use in Propean. They concern the workload, the steps and the resources involved in the scenario considered. For each annotation we specify the associated stereotype, the attributes and the UML extensions (PA attributes) used for representing this domain concepts (for more detail see [36]). In particular:

- **closed workload**: a fixed number of jobs cycles indefinitely in the scenario, and spends an external delay period. The stereotype used is <<PAclosedload>>
  - Attributes and associated PA attributes:
    - *Population:* the size of the workload, i.e., the number of jobs involved (PApopulation)
    - *Response time:* the delay between the instant in which the scenario starts and that in which it is completed (PAresptime).

- **Step:** each increment in the execution of a scenario that can involve the use of resources is a step. The granularity of a step depends on the level of abstraction associated to the scenario. The stereotype used is <<PAstep>>
  - Attributes and associated PA attributes
    - *Repetition:* the number of times the step is repeated (PArep)
    - *HostExecutionDemand:* the total execution demand of the step on its host resources, i.e., the service demand necessary for accomplishing the request (PAdemand)
- **Resource:** This can be passive or active and can participate in one or more scenarios. The former is generally protected by an access mechanism and can represent either a physical device or a logically protected-access. The latter can be a processor, an interface or a storage device and is characterized by the processing steps allocated to it along system deployment. The stereotype used is <<PAresource>>
  - Attributes and associated PA attributes
    - *Utilization:* this is usually the result of an analysis and represents the computed utilization of processing resources expressed as a percentage. For a passive resource in particular it represents the mean number of concurrent users of the resource (PAutilization)
    - *SchedulingPolicy:* the policy that controls the resources, i.e., the rules for assigning the resources to a set of steps (active resource) or the access control policy for handling requests from scenario steps (passive resource). The scheduling policy can be for example FIFO (first-in-first-out), PS (processor sharing), LIFO (last-in-first-out) and so on (PASchdPolicy)
    - *ProcessingRate:* (only for active resources) the relative speed factor of the processor, expressed as a percentage of some normative processor (PArate)
    - *IsPreemtable:* (only for active resources) the possibility for the resource to be preemptable or not, once it starts the execution of an action (PApreemptable)
    - *Throughput*: the rate at which the resource performs its function (PAthroughput)

The numerical values associated to the PA attributes may have different interpretations; for example, they may represent a fixed value, a variable to be estimated, an average value or a distribution, or else they may be a prediction, a measure or a requirement. To model PA value semantics, RT-UML follows a

predefined syntax, whereby it is possible to specify all the desired characteristics (for an example of application see Section 5).

3. CASE STUDY

The proposed methodology can be useful at any stage of development, as soon as the project manager is called to dynamically take the most appropriate decision based on the actual project status and the emerging circumstances. In these situations, performance analysis techniques can help to predict the outcomes that will result from manager's assumptions and to early figure out whether under the current workflow the settled objectives will be met.

In particular, the case study we investigate here concerns the release decision for a software product. The factors that influence this decision can be many, including marketing exigencies, timing constraints, quality requirements, or resources availability. Here we consider that the release follows a test and debug phase, and that the decision is primarily driven by the product quality, measured in terms of found bugs. More precisely, we suppose that, as usual, the testers report each failure found during the test execution in a form, called the trouble report, and that the product will be released only after the testing is completed with no trouble report left open.

We consider that at the beginning of the test phase the manager faces either of two different situations: (i) considering the actual personnel availability, he/she wants to early predict the expected time to release; (ii) for a fixed release time, early agreed on the basis of customer exigencies, he/she wants to decide the most adequate personnel configuration to respect the time constraints. In this paper we illustrate the use of Propean for pursuing either of goals (i) and (ii) considering a simplified case study derived from [14], to which we refer for further detail.

As a first step, we model the organization structure of the company considering the testing stage and the management of reported problems (we disregard the teams not directly involved in these activities). The organization is composed by a project manager PM, a test team T (1÷3 people), a development team D (2÷4 people) and the system architects SA (1÷2 people).

The testers start to execute the planned test cases and every few days (we assume 3 in this example), they insert the trouble reports in an on-line database, called the tracking system TS, which only the testers and the project manager can modify.

At each TS update, the PM analyses the trouble reports and takes the proper resolution for each reported problem. We consider three possible outcomes from his/her analysis:

- The problem must be fixed: the PM classifies the problem as “open” and passes it on to the developers. In this case study for simplicity we assume no prioritization politics among failures, i.e., all reported problems are assigned the same severity (different priorities could also be handled, but the example would be more complicated).
- The problem can be deferred. The PM chooses to leave the problem in the current version of the product and to do the fix in a subsequent release. The problem is classified as “deferred”.
- The problem is not recognized as such. From the trouble report analysis the PM concludes that it is not a real problem, because the program was actually supposed to work in that way. The problem is classified as “as designed”.

The TS update with the problem classification as “deferred” or “as designed” by the PM closes the trouble report (at least for this product release). If instead the problem is classified as “open”, further actions must be taken as described below.

On receiving the open problem reports from the PM, the developers first analyze them to check whether they have enough information to fix the problems or need further explanation from the testers about the failure symptoms. In the latter case, the workflow may include an interaction cycle with the testers. Occasionally, the developers may realize that the fix requires a major design change and inform so the PM, who may require the intervention of the software architects to modify the design, after which the developers modify the code accordingly.

After every problem fix, the testers have to retest the modified parts of the program (regression test). They hence either classify the problem as “closed”, updating consequently the associated trouble report in the TS, or possibly generate further trouble reports containing the new problems found during the test phase.

Given this abstract workflow of the activities and personnel involved, the project manager can periodically analyze the status of the TS in order to:

case i) estimate the expected time at which the product can be released, that is when the TS only contains problem reports classified as “deferred” or “as designed”, i.e., there are no remaining “open” problems;

case ii) derive the most efficient personnel organization for releasing the product within the established time constraints.

In case i), if the estimated release time is too late, for example with respect to market exigencies, the PM has to take the proper corrective actions. For instance, the PM could increase the number of people involved in the development or in the test phase or else decide to pursue a later release date. Alternatively, if the involved personnel are handling several projects contemporaneously, the PM could decide to temporarily divert the people from one or more of the concurrent projects to focus on this one. Similar considerations can be made for case ii).

In both situations, it is very important that the PM can base his/her resolution on a reliable estimate, not on a subjective guess, and that he/she can objectively take into account all the likely combinations of events.

This is the purpose of the methodology presented in the following section: we intend to supply the project manager with a tool that uses performance engineering techniques to:

- predict the release time, also allowing for multi-projects management, i.e., the teams are not dedicated full time to a single project
- evidence (by looking at the personnel rate of utilization) the component that represents the bottleneck and is mainly responsible of the release time delay
- identify the most convenient teams composition in order to ensure the all the projects are released within the deadline agreed with the customer, or within the budget allowance.

4. THE METHODOLOGY

As stated in the introduction, the objective of this research work is to propose sound, reliable solutions to support the manager's decisional process in multi-project management. Our proposal is to apply for this purpose well-known techniques from the field of computer performance engineering, such as Software Performance Engineering (SPE) [27], [28] and queueing networks models [19]. Queuing networks, in fact, are the largest widespread method in performance field.[3]

[3] The results presented in this paper could anyway be obtained via the application of other used approaches, like Petri nets, LQNs or process algebras, by applying the appropriate transformation rules from the UML diagrams to these notations [31]-[35].

4.1 Performance concepts used

The SPE basic concept is the separation of the software model (SM) from its execution environment model (i.e., hardware platform model or machinery model, MM). The SM captures the essential aspects of software behavior; we represent it by means of Execution Graphs (EG). An EG is a graph whose nodes represent software workload components and whose edges represent transfers of control. A software workload component is a set of instructions or procedures performing a specific task. EGs include several types of nodes such as basic, cycle, conditional fork and join nodes. Each node is weighted by use of a demand vector that represents the resource usage of the node (i.e., the demand for each resource).

The MM models the hardware platform and is based on the Extended Queueing Network Model (EQNM) [19]. To specify an EQNM, we need to define: the components (i.e., service centers), the topology (i.e., the connections among centers) and some relevant parameters (such as job classes, job routing among centers, scheduling discipline at service centers, service demand at service centers). Component and topology specification is performed according to the system description, while parameters specification is obtained from information derived by EGs and from knowledge of resource capabilities. Once the EQNM is completely specified, it can be analyzed by use of classical solution techniques (simulation, analytical technique, hybrid simulation [19]) to obtain performance indices such as the mean network response time or the utilization index.

We adapt here performance analysis methods to the purpose of handling personnel multitasking and of optimizing workloads in software project management. We follow the metaphor that:

- *project teams* correspond to the *processing resources* in performance models,
- *project activities* are the *tasks* to be performed within established time intervals.

Using the above SPE concepts, the SM captures the aspects relative to the activity planning, while the MM the ones relative to people (over/under)utilization and distribution.

In particular, we apply a method proposed in [10], for the derivation of performance models based on SPE techniques, starting from a set of UML diagrams. Precisely, the SM is derived from a Sequence Diagram (SD), and the MM from a Deployment Diagram (DD). The method then extracts from these diagrams the main factors affecting system performance and combines them to generate a performance model.

The method, in its original conception, used the standard UML diagrams, with simple ad hoc annotations [10]. As said in the Introduction, we here introduce an improved version of that method, relying on the standard RT-UML profile, to which the derived SD and DD fully comply.

4.2 Propean Basic Steps

We outline the steps of the Propean methodology (and who is in charge of each of them):

1. Manager*: Analysis*

   In this step the project manager defines the project activities under consideration. In particular he/she has to associate to each activity an estimation of the time and the resources necessary for completing it and the role of people involved[4]. In organizations with stable processes, this information can be derived from previous experience on similar projects.

2. Manager*: Modeling*

   The results of analysis in step 1 have to be modeled as RT-UML diagrams. In particular the manager should describe, into one or more SDs, the scenario(s) representing the adopted release process. In the SDs the objects represent the teams involved, and the messages represent the requests of execution of a set of activities or correspond to information/data exchanged between the teams. Moreover the manager should construct a Deployment Diagram (DD) modeling the resources available and their characteristics. In this case the nodes of the DD can be associated to: classical resources (device, processor, database), different project teams, communication means such as, for example, the intranet device. The links between the DD nodes represent the communications among the teams or the documents exchanged inside the organization. We note that the manager has not to repeat this step from scratch each time he/she needs to make estimations about a project. In a mature organization, for similar products, the effort necessary to derive this reference structure will be made only the first time. The same diagrams can then be re-used for successive similar applications, by possibly updating the associated parameters, as will be describe in the next steps.

3. Manager*: Model annotation*

   The two types of diagrams developed at step 2 have to be annotated with the proper values and parameters. The project manager should express, by using a comment-based annotation, the

[4] Note that this is a classical manager duty and not a specific request of the proposed method.

attributes associated with events and actions of the diagrams. In particular, referring to the attributes relative to the PA profile described in section 2.3 the PA attributes of the closed workload will be associated with the first action of the SD; those of the steps will be linked to each of the subsequent message activations; those of the resources will be related to each of the nodes of the Deployment Diagram. In Figures 1 and 2 we report an example of the resulting SD and DD. The details of these figures will be described in the next section.

4. Automatic*: SPE models generation*

   By applying the method proposed in [10], and detailed in Appendix A, it is possible to derive a model for the planned activity (the SM based on EG) and a model for the involved teams and resources (the MM based on EQNM).

5. Automatic*: model evaluation*

   The EQNM obtained in the previous step, which represents the teams and the activities, can be solved to obtain relevant results such as: the predicted completion time for the project (or for a single phase), the resource utilization rate, the best resource distribution with respect to a given completion time, and so on.

6. Manager*: analysis of results*

   The results automatically obtained in step 5 are analyzed by the project manager and, if different from those expected, he/she can go back to step 1 (or 2), make some modifications to the diagrams or to the assigned parameters, and repeat the process until the desired results are obtained.

5. APPLICATION TO THE CASE STUDY

Let us consider for each of the above steps how the method is applied to the case study described in Section 3.

1. Analysis: In this case the project manager decides to focus the analysis on the testing phase. During this step he/she has mainly to define the boundary conditions, such as for example the resources involved and the strategy to adopt for project release, and to establish which parameters (symbolic expressions) are relevant for the estimation, possibly postponing their evaluation to steps 5 and 6.

2. Modeling: during this step the manager develops the SD and the DD representing respectively the sequence of the activities performed during the testing phase and the overall organization of the different teams. These two diagrams are presented respectively in Figures 1 and 2, the meaning of the stereotypes and tagged values of RT-UML will be explained in the next step.

3. Model annotation: The SD and the DD developed must be annotated with the proper parameters.

Considering the SD, the attributes relative to the closed workload are associated to the first action of this diagram. The note must report the name of the stereotype (<<PAClosedLoad>>) followed by the parameters associates to the PA attributes that are:

- PApopulation = *$Nuser* that represents the number of jobs in the scenario: in our context, the number of projects contemporarily under testing. The symbol *$* indicates that *$Nuser* is a variable that the project manager will instantiate with an appropriate value before starting the automatic derivation of the required estimations.
- PAresptime = *('msr','mean', $t_to_release)* that represents the completion time and is one of the expected results. It is modeled as a measured ('*msr*') distribution whose mean is expressed by the variable *$t_to_release*.

The other steps of the SD are annotated with the stereotype <<PAstep>> associated with different PA attributes, depending on the activity considered.

Considering the second and third step of the SD, the testers every tree days (*Nrep*) insert in the database a certain number of trouble reports (denoted as *$N*). The insertion has a mean value equal to *ts*. In other words, $*N* and *ts* are the values to be estimated by the project manager, possibly with the help of testers. They are associated to the PA attributes as follows:

- PArep = *Nrep* number of insertions in the data base
- PAdemand = *('req','mean', ts)* the execution demand of this step on its host resource, i.e., the time necessary ('*req*') to the testers to insert a trouble report into the database, follows a distribution whose mean is given by *ts*.

When the Project Manager analyses the trouble reports, he/she observes a variable number of bugs reported (*$N*). The value to be associated to this variable generally depends on the project typology and can be estimated for a family of similar products. The Manager can classify each bug as "open", "deferred" or "as designed" with a given probability (denoted as *p_fix, p_def, p_des,* respectively*)* and

spending a certain amount of his/her time (denoted as *t_fix_PM, t_def_PM, t_des_PM,* respectively*).* The associated values must be estimated by the Project manager based on his/her experience. Let us consider for example the deferred bugs: the value $*N*p_def* will give the number of bugs classified as deferred among the *$N* reported and *t_def_PM*$N*p_def* will represent the time necessary to the project manager for dealing with them. The described situation is simply modeled by associating to the relative step the stereotype <<PAstep>> with attribute:

- PAdemand = *('req','mean', k)* where k can assume values *t_fix_PM*$N*p_fix, t_def_PM*$N*p_def, t_des_PM*$N*p_des* depending on the considered SD step. It represents the time necessary ('*req'*) to the project manager to deal with the different trouble reports and follows a distribution whose mean is given by *k*.

Similar consideration can be done also for the others steps of the SD, annotating each step with the parameters of interest and estimating the required values.

Considering the DD, its nodes can refer to both classical resources (device, processor, database) and people teams. Moreover, the DD models also the communication nodes: for instance, the Intranet to access the database TS and a meeting room symbolizing a "communication channel" among different teams. Each node represents a kind of resource and therefore it is necessary to associate to each resource a stereotype <<PAhost>>. Then, depending on the resource considered, the associated PA attributes are:

- PAschdPolicy=*P* where *P* can be equal to *FIFO*, *PS* or *PR* and models the strategy by which the resource handles the different jobs.
- PApreemptable= *Yes* In the case study it is supposed that only the System Architect team can be interrupted during his/her work.
- PAutilization=*$Util* represents the rate of utilization of the different resources. The value associated to this variable is an analysis result.
- PArate=1 the resource works full time on the assigned job.
- Pathroughput=*Np* it represents the amount of work provided per unit of time (a day in the case study) by a person belonging to a certain team. This value is normalized to 1 in case of a single person, 2 when two people work together, and so on.

4. SPE model generation: Following the steps described in the appendix, from the annotated SD and DD the corresponding EG and EQNM can be automatically derived. Figures 3 and 4 illustrate respectively the high level EG and one of the low level EGs obtained from Figures 1 and 2, while Figure 5 shows the EQNM with a team composition made of: 1 project manager, 1 software architect, 1 tester and 2 developers (1PM, 1SA, 1T, 2D).

With respect to the SD and the DD, in this step we have made the following choices:

i. the database TS and the connected Intranet have not been modeled, because the times involved in the TS accesses are orders of magnitude less than the times required by the activity steps (msecs vs. days);

ii. the meeting room has been introduced as a delay center modeling the communications with the manager.

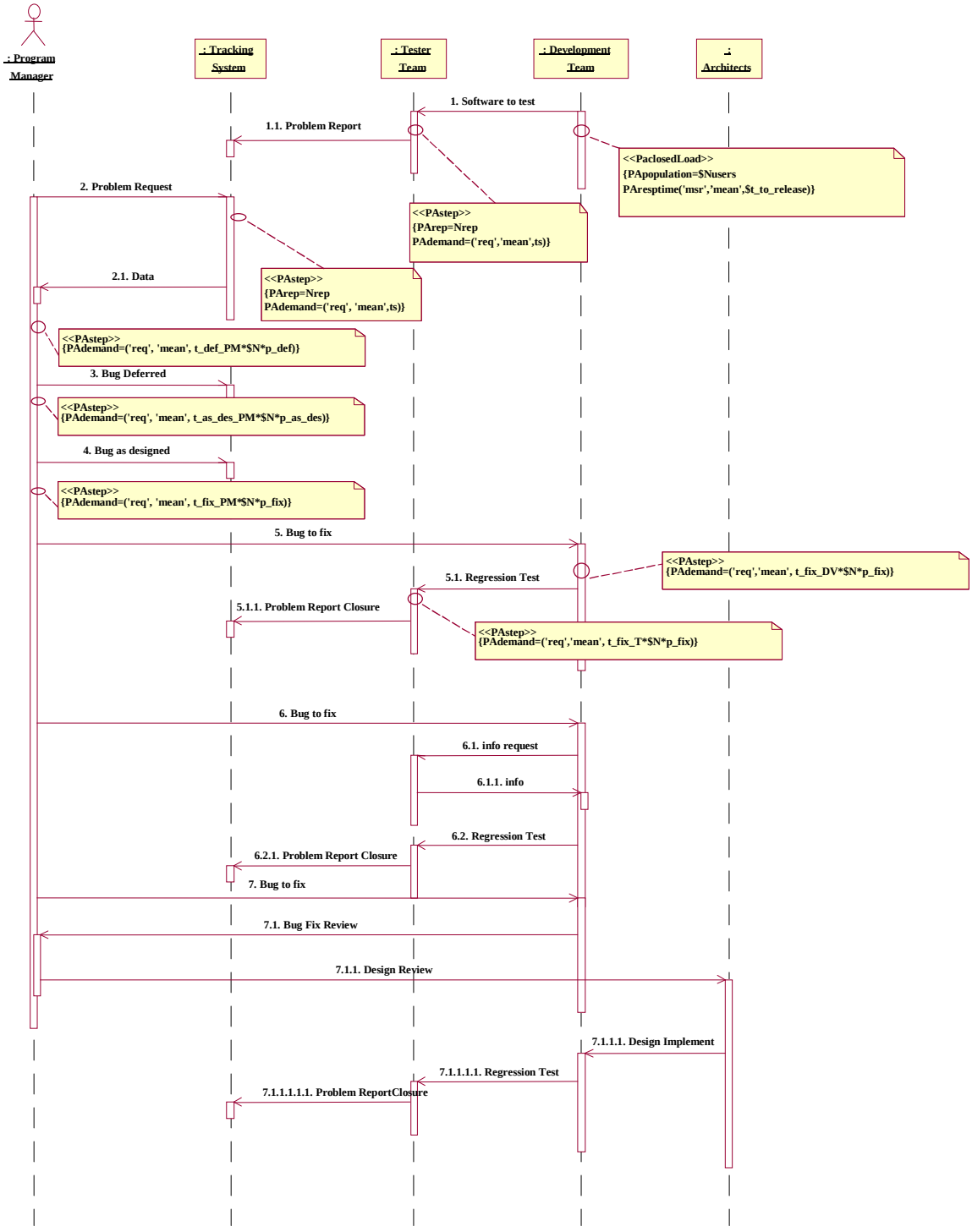

Figure 1: Sequence Diagram

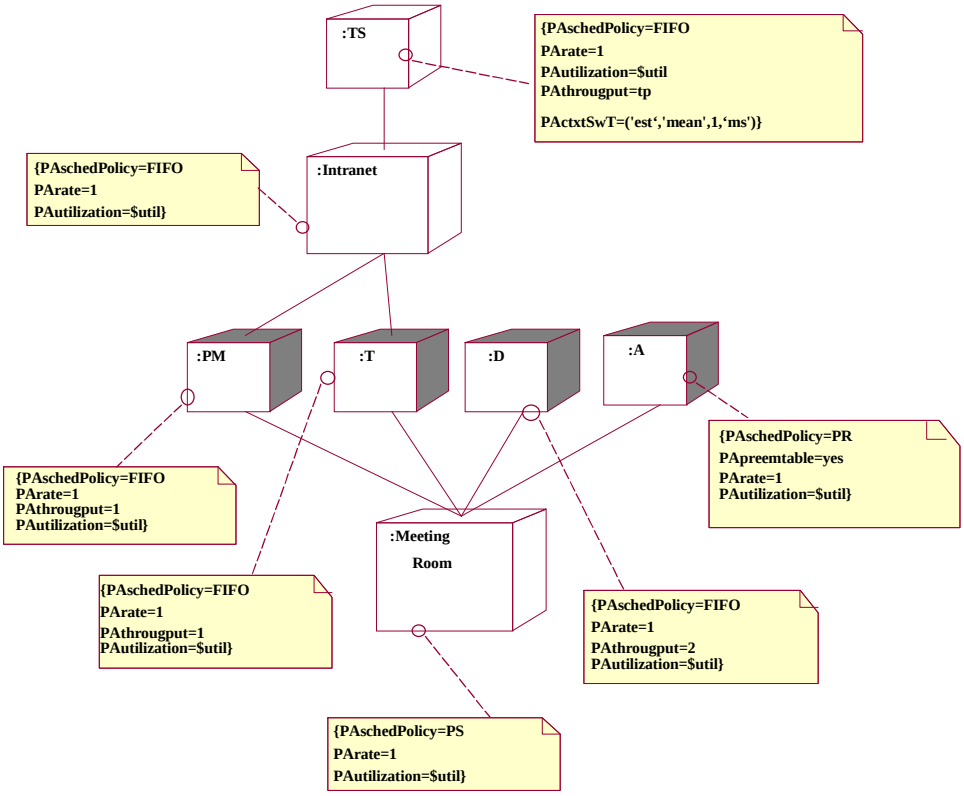

Figure 2: Deployment Diagram

Figure 3 represents an EG at a high level of abstraction modeling the main activities of the testing phase without details, while Figure 4 shows the details of the block named "problem analysis", by illustrating several activities modeled in Figure 1. Moreover, the demand vector for each block is derived by combining information coming from annotations in the SD and in the DD.

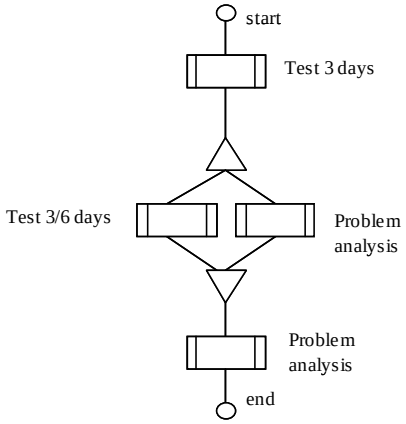

Figure 3: High level EG obtained from SD in Figure 1

For example, the first block is called “3+4” because it models the interactions 3 and 4 in the SD; its associated demand vector represents the service demand to the resources involved in the scenario for the management of bugs that are deferred or classified “as designed”. In such a case only the manager is involved and his/her service demand can be derived from the annotated SD as

(t_def_PM* $N*p_def + t_as_des_PM*$N*p_as_des).

Note that the different kind of projects (depending, for example, on the test duration or on the number of bugs) generate different instances of the demand vectors for the EGs in Figures 3 and 4, and therefore different routing chains in the EQNM. The possible choices lead to generate different models to be evaluated in the next step.

5. Model evaluation: several analyses can be done by assigning different values to parameters in the EQNM. Examples of various model evaluations are illustrated in the next section.

6. Analysis of results: the manager can make several kinds of decision by analyzing the results obtained in the previous step. Again, an example of this analysis is illustrated in the next section.

5.1 Discussion

It is important to point out that on the manager’s side the effort required to employ the methodology is to explicitly derive in a SD (such as the one shown in Figure 1) a high level model of the activity workflow and in a DD (as in Figure 2) the organization structure. He/she does not need to know all the other details on how such models are then translated into SPE models and then solved (and that we described above for completeness).

We understand that even the derivation of the RT-UML diagrams could be felt at first impact as an undesirable extra burden for the already overloaded manager. However objectively it should not be much effort: if one has a clear view (as plausibly the manager must have) of how the development process is structured and of which are the activities to accomplish and their mutual influences, deriving the RT-UML diagrams that depict them at a high level of detail should not take much labor, especially with the support of an appropriate interactive tool. Besides, we expect that the returns make it worthwhile.

In fact, once such diagrams have been derived, various interesting analyses can be conducted in completely automated way. The manager can make different assumptions on the parameters of the modeled scenarios and obtain automatically a reliable prediction of what will be the outcomes consequent to each assumption.

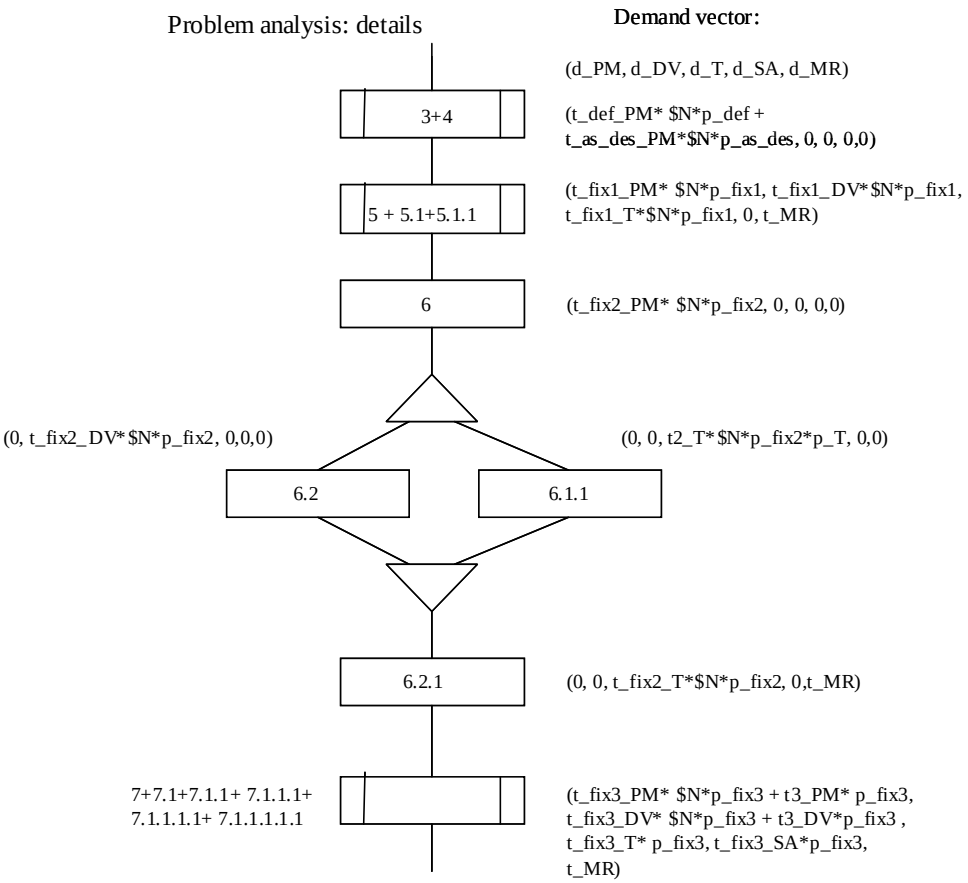

Figure 4:The low level EG for "problem analysis" block obtained from SD in Figure 1

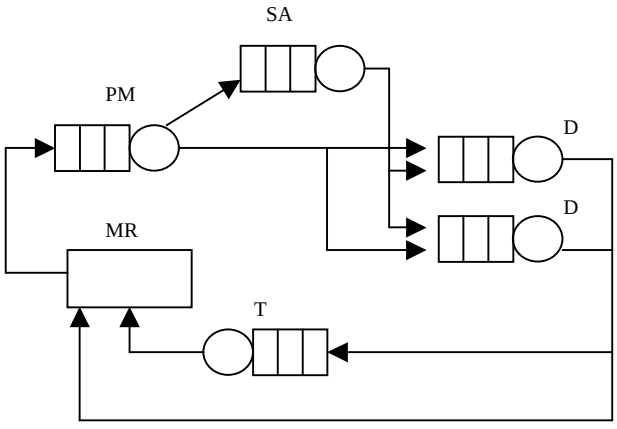

Figure 5: The EQN Model obtained from SD in Figure 1 and from DD in Figure 2

6. ANALYSIS OF RESULTS

We present in the following two orthogonal applications of Propean to the described case study, illustrating for each of them the parameters to be introduced and the kind of estimations that can be derived. We consider the two situations in which: (i) the manager wants to predict the completion time of the testing phase (section 6.1); or (ii) he/she wants to derive the most efficient personnel distribution for completing the testing phase within a fixed time deadline (section 6.2).

Generally for each diagram several parameters can be varied, depending on the desired prediction. We have considered the following parameters: the estimated duration of the test period, the number of registered trouble reports, the composition of the involved teams, and whether they are fully dedicated to the examined project or instead are contemporaneously handling other projects.

6.1 Estimating the completion time

Considering the case study described in section 3, we illustrate some plausible situations: in Propean, each different situation corresponds to a variation in the parameter values in SD and DD. For example a possible situation could be represented by the following assumptions:

- the planned duration for the test phase of a given product is six days (considering the attribute PArep in the second step of the SD, the parameter *Nrep* is set equal to 2);
- For the typology of project under test, based on his/her experience, the manager assumes that the number of trouble reports issued will be equal to 10 (considering the attribute PAdemand in the third step of the SD, the parameter *$N* is set equal to 10)
- the personnel in charge for the test and debug phase consists of one tester, two developers and one software architect (plus of course the manager): this configuration is denoted as 1PM, 1SA, 1T, 2D; (as illustrated in the considered DD)
- the tester and the two developers are in parallel engaged in another project (the parameter *$Nuser* of the SD is initialized to 1).

For each parameter configuration, using Propean the manager can thus obtain, early in advance of the testing phase, a prediction of the time in which the product will be ready for release (i.e., no more open trouble reports exist).

We report in Table 1 the obtained results for different parameters values. In the table the estimated time to release is measured in days, considering one working day equal to 8 hours (optimistic bound), and the results are rounded to the closest integer. The table shows the release time when the planned duration for the test and debug phase is three, six or nine days (with a group of four columns for each case), and when 2, 10, or 20 trouble reports are issued, as indicated in each row.

For each case, then, we derive the estimate when the resources (the people) are fully dedicated to the project under exam (denoted as T&D Full Ded); the test team is fully dedicated, while the developers are handling this and another project (D Shared 1); both the testers and the developers are handling this and another project (T&D Shared 1), and finally both the testers and the developers are handling three more projects in addition to this one (T&D Shared 3).

Table 1. Estimated time to release in days

		Planned Test Duration=3 days				**Planned Test Duration=6 days**				**Planned Test Duration=9 days**			
	Project #Bugs	**T&D Full Ded**	**D Shared 1**	**T&D Shared 1**	**T&D Shared 3**	**T&D Full Ded**	**D Shared 1**	**T&D Shared 1**	**T&D Shared 3**	**T&D Full Ded**	**D Shared 1**	**T&D Shared 1**	**T&D Shared 3**
1PM 1SA 1T 2D	**2**	5	6	8	13	8	10	14	25	11	13	17	28
	10	9	10	11	17	12	13	17	29	14	16	20	31
	20	14	15	16	22	16	18	22	32	18	20	24	35

Going back to the situation described above, in which the test duration was assumed equal to six days and the number of trouble reports to 10, the time necessary to complete the testing phase is estimated into 17 days from the start of the test phase (2^{nd} row, 7^{th} column). If the manager was pointing towards a much earlier release deadline, Propean shows it is unlikely that he/she will be able to meet it. Let us assume that the target release deadline is 12 days. Even considering the more optimistic hypothesis that only 2 bugs are encountered, in the present configuration the release time would be not shorter than 14 days (1^{st} row, 7^{th} column). Thus, either the manager can accept a more relaxed deadline, or he/she takes some countermeasure. In particular, if the project under exam has high priority, a possible solution could be to take away from the other parallel project the resources (the tester and the developers) that are necessary to complete this one. In this case if they are fully assigned to the completion of the testing phase of this project, then the predicted release time with 10 bugs is reduced to 12 days (2^{nd} row, 5^{th}

column), which was the target deadline. Thus, by means of simple SPE analyses, the manager gets statistical predictions that can support his/her decisional process.

On the other hand, adopting the last solution results in an increased project cost, due to the under-utilization of certain personnel categories. Another relevant parameter the manager should consider before taking any decision is in fact the rate of utilization of the involved teams. This analysis is automatically obtained with the parameter assignments used for the estimation of completion time and can be very useful not only to better administrate human resources, but also to identify the bottlenecks when a phase takes too long.

In Table 2 we report the percentage of the utilization rate, denoted by ρ, for the tester and the developers considering the same parameters assignment of Table 1. This index is measured by the ratio between the frequency at which requests arrive, and the frequency at which the processing element (in our case a team) can deliver services. The utilization rate varies between 0 and 1, where 1 means that the resource is saturated, and can represent a bottleneck; 0 means it is idle, and a good utilization is somewhere in the middle.

In the initial configuration we assumed one tester and two developers, employed in this and in another project. We can see that the bottleneck is clearly the tester, as the utilization rate percentage is computed as 99%, while the two developers are well employed, with a rate of 55%. Deciding to fully dedicate one tester and two developers to the test and debug phase allows the manager to meet the deadline, but in such a configuration the developers are under-utilized, at 29%.

One further possibility to explore could be to devote one tester at full time, while leaving the two developers on both projects. In such a configuration we would get a release period of 13 days, but the resources are better employed (46% the tester and 58% the two developers).

Table 2. Percentage of utilization rate of testers and developers

		Planned Test Duration=3 days								**Planned Test Duration=6 days**								**Planned Test Duration=9 days**							
	Project	**T&D Full Ded**		**D Shared 1**		**T&D Shared 1**		**T&D Shared 3**		**T&D Full Ded**		**D Shared 1**		**T&D Shared 1**		**T&D Shared 3**		**T&D Full Ded**		**D Shared 1**		**T&D Shared 1**		**T&D Shared 3**	
	#Bugs	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D
1PM 1SA 1T 2D	**2**	21	23	15	59	99	55	100	77	66	10	50	56	99	53	100	76	79	70	65	54	99	52	100	75
	10	29	29	25	57	99	56	100	74	56	29	46	58	99	55	100	76	69	16	59	56	99	54	100	76
	20	31	31	28	55	99	55	100	73	49	30	46	56	99	55	100	74	69	23	56	55	99	55	100	76

Analyzing the results in Table 1 another interesting fact can be observed: although obviously the duration of the test and debug process can be highly influenced by the number of bugs found, a rational organization of the personnel is more crucial, especially for large enterprises dealing with more development processes in parallel. The release delay in fact increases faster as the teams get involved in more contemporaneous projects than if we increase the estimated number of bugs.

For instance, considering a large product with a planned test period of 9 days, when all the resources are fully dedicated the expected time to release, even foreseeing 20 bugs, is 18 days, against the 20 days estimated to handle half (i.e., 10) bugs if the tester and the developers are contemporaneously employed in another project. If we further consider a configuration in which the tester and the developers are handling three more projects, even though in this project we optimistically assume to find only 2 bugs, handling them would take 28 days.

Another possible countermeasure when the predicted release time exceeds the target deadline is to add more personnel to the development. Using Propean, revising the estimates is immediate and again it consists only in changing some of the configuration parameters.

Let us consider, for example, that the personnel in charge for the test and debug phase consists of two testers, two developers and one software architect, plus of course the PM. This configuration is denoted as PM1, SA1, T2, D2. We report, in Tables 3 and 4 respectively, the estimated time to release and the utilization rate of the testers and developers.

Considering the initial situation in which the test duration was equal to six day, the number of trouble reports to 10 and the committed release time 12 days, even if one more tester is added, the product would be ready in 15 days (as reported in Table 3, 2^{nd} row, 7^{th} column), instead of 17 as with the previous configuration, but this could not be sufficient. In this new configuration, in fact, the manager would be able to meet the target deadline only if the estimated number of bugs is 2.

The utilization rate of testers with 10 bugs (table 4) is equal to 55% (instead of 99% of the previous case). This means that personnel organization in this case is better than before and the tester resource is not anymore the bottleneck of the development process. As evidenced by the table, in this case assigning the testers and the developers full-time to the project, or even only the developers, would be

meaningless. Their utilization rates in the two cases, (30% and 22%) and (24% and 31%) respectively, in fact, evidence an under-utilization of the resources.

Table 3. Estimated time to release in days

		Planned Test Duration=3 days				Planned Test Duration=6 days				Planned Test Duration=9 days			
	Project #Bugs	T&D Full Ded	D Shared 1	T&D Shared 1	T&D Shared 3	T&D Full Ded	D Shared 1	T&D Shared 1	T&D Shared 3	T&D Full Ded	D Shared 1	T&D Shared 1	T&D Shared 3
1PM 1SA 2T 2D	**2**	5	6	6	10	8	10	12	19	11	13	14	22
	10	9	10	11	14	11	13	15	22	14	15	17	24
	20	14	16	16	20	16	17	20	27	18	19	21	28

Table 4. Utilization rate of testers and developers

		Planned Test Duration=3 days								Planned Test Duration=6 days								Planned Test Duration=9 days							
	Project	T&D Full Ded		D Shared 1		T&D Shared 1		T&D Shared 3		T&D Full Ded		D Shared 1		T&D Shared 1		T&D Shared 3		T&D Full Ded		D Shared 1		T&D Shared 1		T&D Shared 3	
	#Bugs	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D	T	D
1PM 1SA 2T 2D	**2**	12	24	7	59	55	58	76	77	37	10	26	29	55	55	74	75	40	70	34	54	55	54	70	76
	10	14	29	13	57	55	56	76	75	30	22	24	31	56	56	75	75	36	17	33	56	56	55	72	76
	20	15	30	14	55	54	55	75	70	26	27	22	31	56	57	74	74	33	24	28	55	56	55	74	74

6.2 Deriving the best personnel distribution

As in the previous section we discuss some situations for illustration purposes. Considering the case study, we report in this section the results obtained under the following assumptions:

- the planned duration for the test phase of a given product is 3 days, (considering the attributes PArep in the second step of the SD the parameter *Nrep* is set equal to one);
- For the typology of project under test the manager assumes that the number of trouble reports issued will be equal to 2 (considering the attributes PAdemand in the third step of the SD the parameter *$N* is set equal to two);
- The personnel in charge for the test and debug phase consists of one software architect, the program manager, while the number of testers and developers is variable. The configuration is denoted as

1PM, 1SA, T*t*, D*d* where the variables *t* and *d* indicate the values to be established by using Propean.

More precisely, the goal of the manager, given the above parameter assignment, is to define the values to assign to the variables *t* and *d*, i.e. the best personnel assignment, so that the project can be released within no more than seven days (considering one working day equal to 8 hours, and the results rounded to the closest integer).

Table 5 reports some of the obtained results when the planned duration for the test and debug phase is three days, and considering the testers and developers fully dedicated to the project under exam (denoted as T&D Full Ded); the testers fully dedicated, while the developers are handling this and another project (D Shared 1); both the testers and the developers are handling this and another project (T&D Shared 1), and finally both the testers and the developers are handling three more projects in addition to this one (T&D Shared 3).

In particular we suppose that when a team (resource) receives the request of a job, the task is performed by only one of the people available in that moment.

Table 5. Estimated completion time at various configurations

Configuration 1PM, 1SA, T*t*, D*d*		**Planned Test Duration=3 days**			
		T&D Full Ded	**D Shared 1**	**T&D Shared 1**	**T&D Shared 3**
#Bug 2	***t=1, d=2***	5	6	8	13
	t=2, d=2	5	6	7	10
	t=3, d=4	5	5	6	8
#Bug 10	***t=1, d=2***	9	10	11	17
	t=2, d=2	9	10	11	14
	t=3, d=4	9	9	10	12
#Bug 20	***t=1, d=2***	14	15	16	22
	t=2, d=2	14	15	16	20
	t=3, d=4	14	15	15	17

Therefore if the estimated number of trouble reports is equal to 2 and the target release date is 7 days, from the analysis of Table 5, a good configuration could be a tester and two developers completely

dedicated to this project, one tester and two developers with developers handling this and another projects, or, alternatively, two testers and two developers engaged at the same time in another project. The other configurations can be immediately excluded because beyond the deadline.

The manager can also derive a more precise evaluation of the cost of project realization, by using the utilization rate of the people involved. We report for this reason in Table 6 the corresponding percentage of the utilization rate of the tester and developers, while a more thorough discussion of the project cost is deferred to Section 6.3.

Table 6. Utilization rate (%) of testers and developers

Configuration 1PM, 1SA, T*t*, D*d*		**Planned Test Duration=3 days**							
		T&D Full Ded		**D Shared 1**		**T&D Shared 1**		**T&D Shared 3**	
		T	D	T	D	T	D	T	D
#Bug 2	***t=1, d=2***	24	24	15	60	99	56	100	76
	t=2, d=2	12	24	0.7	60	55	58	76	77
	t=3, d=4	0.7	12	0.6	33	38	30	60	52
#Bug 10	***t=1, d=2***	30	29	25	57	99	55	100	74
	t=2, d=2	15	29	12	57	55	56	76	75
	t=3, d=4	10	15	0.9	30	38	30	60	50
#Bug 20	***t=1, d=2***	31	30	27	55	99	55	100	75
	t=2, d=2	16	30	14	54	55	56	76	76
	t=3, d=4	10	15	0.9	30	38	30	61	50

An alternative situation is that when a team (resource) receives the request of a job, the task is performed by all the people available in that moment. This means that if two people are available, they will work in parallel for completing the job. Tables 7 and 8 report the estimated completion time for the different configurations and the percentage of utilization rate of the testers and developers. The different policy of job completion is incorporated by the resulting utilization rates of the testers and developers.

Table 7. Estimated completion time at various configurations

Configuration 1PM, 1SA, T*t*, D*d*		**Planned Test Duration=3 days**			
		T&D Full Ded	**D Shared 1**	**T&D Shared 1**	**T&D Shared 3**
#Bug 2	***t=1, d=2***	5	7	8	15
	t=2, d=2	4	6	8	14
	t=3, d=4	4	5	6	8
#Bug 10	***t=1, d=2***	7	10	11	17
	t=2, d=2	7	9	11	16
	t=3, d=4	6	7	8	10
#Bug 20	***t=1, d=2***	12	14	15	21
	t=2, d=2	11	12	13	19
	t=3, d=4	9	10	10	13

Table 8. Utilization rate of testers and developers

Configuration 1PM, 1SA, T*t*, D*d*		**Planned Test Duration=3 days**							
		T&D Full Ded		**D Shared 1**		**T&D Shared 1**		**T&D Shared 3**	
		T	D	T	D	T	D	T	D
#Bug 2	***t=1, d=2***	26	28	13	59	99	56	100	77
	t=2, d=2	14	29	0.6	60	52	57	76	77
	t=3, d=4	11	18	0.6	35	38	32	61	53
#Bug 10	***t=1, d=2***	34	35	26	62	99	60	100	78
	t=2, d=2	19	38	13	64	56	60	77	78
	t=3, d=4	16	25	14	40	42	38	65	56
#Bug 20	***t=1, d=2***	36	37	31	62	99	61	100	78
	t=2, d=2	20	40	17	63	56	62	77	78
	t=3, d=4	17	26	16	40	42	39	63	54

6.3 Cost estimation

Achieving the target deadline is certainly important. However, another relevant factor that the manager must also consider is the associated cost. For each selected configuration we can derive a rough estimation of cost denoted as $C_G(i)$, i=1, …, N_c (total number of configurations), and computed as follows:

$$C_G(i) = d(i) * \sum_{k \in \{T,D\}} [(p_k(i) * c_k)/s_k(i)]$$

Where : d (i) denotes the working days for configuration i

$p_k(i)$ is the number of people in team k

c_k is the cost associated to each person in team k

$s_k(i)$ is the number of projects shared by team k in configuration i

Let us compare, for example, the configurations *a* and *b* $(a,b \in \{1,\ldots, N_c\})$, where

- *a* corresponds to: 6 days of planned test duration, 10 number of bugs, T&D shared1, 1PM, 1SA, 1T and 2D
- *b* corresponds to: 6 days of planned test duration, 10 number of bugs, T&D shared1, 1PM, 1SA, 2T and 2D.

$$C_G(a) = d(a) * \sum_{k \in \{T,D\}} [(p_k(a) * c_k)/s_k(a)] = 17 [(1*c_T)/2 + (2*c_D)/2]$$

$$C_G(b) = d(b) * \sum_{k \in \{T,D\}} [(p_k(b) * c_k)/s_k(b)] = 15 [(2*c_T)/2 + (2*c_D)/2]$$

Let us suppose, for the sake of simplicity that $c_T = c_D = c$, we obtain:

$C_G(a) = 17*c*3/2 = 25.5*c$

$C_G(b) = 15 *c*2 = 30 *c$

The manager therefore should decide by comparing configuration *a* and *b*, whether the increase of the project cost is justified by a reduction of only two days in the completion time.

Another information that can aid the manager in decision-making is the computation, basing on utilization rate, of a so-called “waste” factor, that is the cost of people under-utilization, as follows:

$W_G(i) = d(i) * \sum_{k \in \{T,D\}} [(p_k(i) * c_k*(1-\rho_k))/s_k(a)]$

Therefore for configuration *a* and *b* above, we can compute

$W_G(a) = 17 * [(1 * c*(0.01))+(2*c*0.45)/2]=17*c*0.46=7.82*c$

$W_G(b) = 15 * [(2 * c*(0.44)/2)+(2*c*0.44)/2]=15*c*0.88=13.2*c$

Note that, by use of the attribute PArate we can give also an idea of the expertise of people belonging to the team, with PArate $\in$ [0,1], where 1 denotes a good skilled person, while 0 denotes a beginner. So, we can weight the cost c_k by use of PArate to take into account the cost of different people. Obviously also the other parameters such as the center service time modelling people work should be weighted by the PArate value. In the examples above we have supposed, for simplicity, PArate equal to one.

Note that currently, due to restrictions imposed by the PA profile, it is necessary to suppose that all people in the same team have the same skill and cost (an average value can be taken, if this is not the case).

7. ARCHITECTURE OF THE PROPEAN TOOL

As already stated, the final goal of this research work is an automated environment that the manager can easily consult in his/her everyday activity to get advice for sound decision-making. With reference to the stepwise procedure described in Section 4.2, this environment should incorporate a tool fully automating steps 4 and 5 (the SPE related computations), and provide as well support to the other steps pertaining to the manager, facilitating the RT-UML modeling of the workflow and the resources according to the required formats. Currently we have already available some small pieces of such a tool (which we used to process the case study presented here), while further implementation work is ongoing to complete the platform. In this section we overview the architecture of the Propean tool. To make the SPE calculations, Propean transforms the UML model annotated with performance information into an Extended Queueing Network performance model [19]. Following ([36], cap10) the input to our transformation algorithm is a file that contains an annotated UML model translated in XML format according to the standard XMI interface [36], and the output is the corresponding EQNM model description file, which can be read directly by existing EQNM solvers [19]. The tool architecture is illustrated in figure 6. A UML tool (such as Poseidon [41]) processes the input diagrams and generates the XML file. The UML to EQNM transformation component takes in input the XML file and produces

as an output a file describing the performance model. Specifically, the EQNM model structure is generated from the high-level software architecture described in the DD indicating the allocation of software components (in our cases the activities) to hardware devices (in our case the teams). The EQNM model parameters are obtained from detailed models of key performance scenarios, represented in the SDs. The derived performance model goes to a performance model solver (EQNM analytical solver or simulator) that derives the performance analysis results. Finally the Results Convertor analyses the performance results and convert them back in the UML model as constraints on some *PAattribute*, thus completing a round-trip tour.

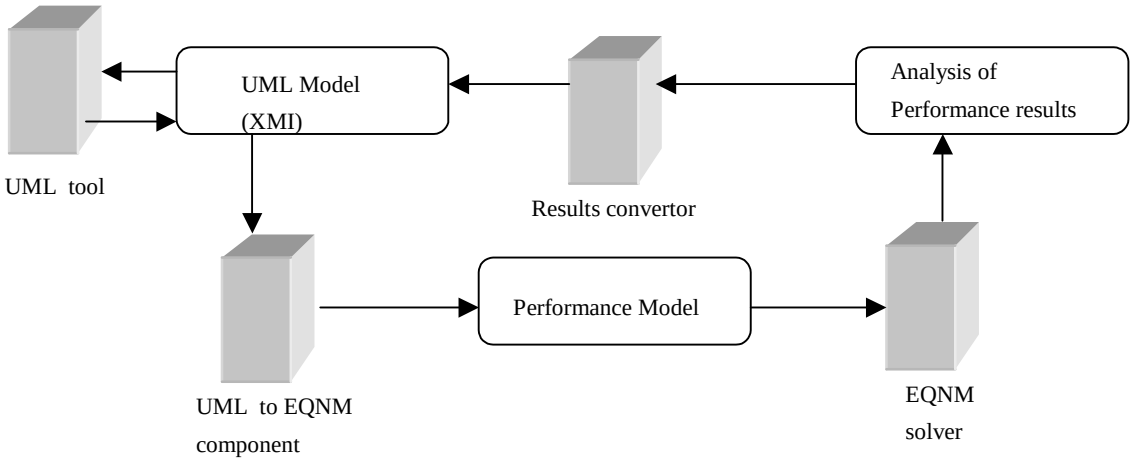

Figure 6: Propean Tool Architecture

The policy we are pursuing in the implementation is that, where available, we use existing tools, and integrate them into the Propean tool. For instance, we do not obviously want to develop a new UML tool. To this regard, we notice that the release of the RT-UML profile has happened very recently and there are not yet commercial tools specifically handling it or supporting the Performance profile. Therefore, the XML files must be processed to “attach” the tagged values associated with the stereotypes to different model elements.

8. RELATED WORK

A voluminous literature about project management and development can be found, but little of it treats the problem of multiprojects planning and people multitasking on several parallel projects. We report here a brief survey of previous related studies (we refer to [16] for a more complete review of the literature) and of the more widespread decisional tools.

8.1 Related studies

Two crucial aspects of project management during development are resources distribution and activity planning. These issues belong to a more general research field that is Concurrent Engineering (CE) [29]. This discipline became popular with the studies of Imai et al. [13] and Takeuchi and Nonaka [30] and has greatly influenced both the academic and the industrial approaches to production. However, these works focus mainly in organizing the tasks within a single project, taking into account the decomposition of a complex product design into smaller activities and their subsequent coordination.

Considering the distribution of resources in a multiproject environment, PERT (Project Evaluation and Review Technique) and CPM (Critical Path Methods) [11] are probably the first proposed methods. They describe an idealized flow of project activities, in which no new project is introduced over time and activity durations are treated as deterministic. Markov chain models [16], [38], which assume an activity time exponentially distributed and use matrix methods for deciding the task time order in development [6] were the natural subsequent evolutions.

The work presented here is close to Adler et al.'s [1]. These authors in fact study the problem of personnel organization and resources distribution among several projects developed at the same time, and like us use queueing networks and stochastic processing network models to represent product development and identify the bottlenecks in task scheduling. The authors however focus on five basic process elements: jobs, tasks, procedure constraints, resources, and flow management control. In particular, a single process may need to handle a variety of job types, which in turn are divided in tasks (i.e., activities or operations). Tasks are connected by precedence relations. The resources are engineers and technicians, who are the units that execute the tasks. The flow management control represents how the resources executed a job's constituent tasks. Lock [20] identifies a sixth element consisting of the assessment of individual contributions.

Recently queuing theory has been applied to model software maintenance requests [24] and to management planning [2]. Specifically, in this last case, it is presented a queueing based approach for staffing process management and evaluating service levels. The nodes of a multi-stage, multi-center queueing model are associated to the different maintenance phases. Each stage is considered in series and each entering request is subjected to a sequence of activities before leaving the system.

8.2 Decisional tools

The decisional support managers can use generally is of two kinds. One consists in techniques or methods that visualize resources and personnel and distribute them among the phases of project development. Example are represented by the traditional Control Charts or Gantt Charts [6], or the more innovative Design Structure Matrix (DSM) [8] which can display the interactions between different teams with the process activities. Tools may support these methods, which are extremely intuitive, but generally the validity of the plans depends strictly on the subjective skill of the managers. Besides, the use of these techniques in a multiproject context could be rather difficult.

The second kind of decisional support consists of specialized tools for managers. Microsoft Project tool [39] or the Kerzner Project Management Maturity Online Assessment tool [40] represent some examples of specific tool , which provide a valid help for maintaining an updated database of the available people and resources, and for producing and visualizing a project plan.

Recently the idea of readapting existing tools for management purpose is become more common also for economic aspects and some proposals can be found in literature. An example is the work of Dickinson et al. [12], which shows how to use Dependency Matrix in combination with the existing Portfolio tools to support decisional process, analyze the interdependences between projects and combine them together. Another solution is presented in [7] where the authors propose a tool for production management optimization that uses Gantt Charts and PERT diagrams for visualizing the obtained results.

However, most existing tools consider only a specific aspect of management, focusing for example either on the completion time or on the personnel distribution and, more importantly, they cannot explicitly manage several contemporaneous projects. Finally, the majority of available tools apply ad hoc algorithms for simulating project evolution, based on some parameters values introduced by the user. Some of those tools generate approximated predictions without any guarantee of statistical significance.

9. CONCLUSIONS

In this paper we have discussed the usefulness of applying classical performance analysis techniques to support the management of people and workflows in software processes. We have introduced the Propean approach as an aid for decision-making substantiated by rigorous statistical predictions of time and efforts. Propean input models are derived from RT-UML diagrams depicting the flow of activities and the structure of the enterprise organization.

We have illustrated in a case study encompassing the test and debug phase of a development process how the proposed methodology could be of help to a manager in either of two situations: for establishing a reliable release date (while keeping under control the effort charged to personnel) or for deciding the most effective organization of involved teams when the release date is fixed.

To use the Propean methodology only the knowledge of a small subset of the RT-UML profile is requested on the manager's side, and a limited effort is needed to develop the SD and DD diagrams. All the necessary transformations and desired analyses can then be conducted in automated way by performance model-solving tools.

In evaluating the required effort for Propean application, it must be considered that the models have not to be re-derived from scratch at each application of the methodology. It is in fact plausible to hypothesize that the processes that govern the various phases of development in a company are standardized and do not change completely at each new project. Therefore, after an initial investment to model the various workflows encompassed by the development process in use, at each next application of the methodology the manager only needs to tune the parameters, or in the worst case to make some update to the existing diagrams.

While we hope this paper has provided an evidence of the potential usefulness of performance analysis in project management, we remain aware of course that no tool can replace the expertise and intuition of good managers. We propose Propean as an assistant and certainly not as a substitute for managers. We claim that -if well-used- Propean can alleviate the intellectual burden of trying to account for all potential scenarios and circumstances when making predictions, and can certainly provide a valid contribution to more effective decision-making. The real validity of Propean comes out in complex situations such as the case of multiple parallel projects and of personnel multitasking, while in simple cases it could be an oversized method.

Our future work will include: the investigation of further features of the RT-UML profile, such as for instance the usage of activity diagrams instead of the sequence diagrams; the development of other case studies with their relative RT-UML input models, so that Propean blueprints for various plausible contexts in project management are already available: currently, we are working to Propean models of the whole RUP (Rational Unified Process); finally, of course we are still working on the automation of Propean. We have outlined the architecture of an automated framework that is currently under development.

REFERENCES

[1] Adler. P., S., Mandelbaum, A., Nguyen, V., Schwerer, E. From Project to Process Management: An Empirically Based Framework for Analyzing Product Development Time. Management Science, (Vol. 42, 1995), 458-484.

[2] Antoniol G., Casazza G., Di Lucca G.A., Di Penta M., Rago F. A Queue Theory-Based Approach to Staff Software Maintenance Centers.In Proceedings of IEEE International Conference on Software Maintenance, ICSM 2001, 6-10 November 2001, Firenze, Italy.

[3] Basanieri F., Bertolino A., Marchetti, E., Mirandola, R. Automating the Management of Teams and Tasks in Software Multiprojects using UML and Queueing Networks. Proceedings of 3rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD, Madrid, Spain, June 2002

[4] Bertolino A., Marchetti, E., Mirandola, R. Real-Time UML-based Performance Engineering to Aid Manager's Decisions in Multi-project Planning" in Proceedings of Third ACM International Workshop on Software and Performance, WOSP 2002, Rome, Italy, July, 2002.

[5] Bertolino A., Lombardi G., Marchetti E., Mirandola R. Performance Analysis of the Rational Unified<<Process Product>>, In Proceedings of 12-th International Workshop on Software Measurement, IWSM 2002, Magdeburg, Germany, October 2002

[6] Black, T., A., Fine, C., H., and Sachs, E., M., A Method for Systems Design Using Precedence Relationships: An Application to Automotive Brake Systems. M.I.T. Sloan School of Management, Cambridge, MA, Working Paper no. 3208, 1990.

[7] Bori S., Lores J. Pascual R. Roures E. PROMAN, Planning Production Management and Control System with Intelligent Interface and Advanced Forecast. In Proceedings of ETFA 2001, 8-th IEEE International Conference on Emerging technologies and Factory Automation, Antibes (France), 15-18 October 2001.

[8] Browning T. Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Direction. IEEE Transaction on Engineering Management (Vol. 48, 2001), 292-306.

[9] Burr, A. and Owen, M. Statistical Method for Software Quality: Using Metrics for Process Improvement. Int. Thomson Computer Press, 1996.

[10] Cortellessa, V. and Mirandola, R. PRIMA-UML: a Performance Validation Incremental Methodology on Early UML Diagrams, Science of Computer Programming, 44 (2002), 101-129, July 2002, Elsevier Science

[11] Dean, B., V., Project Management: Methods and Studies, North-Holland, Amsterdam 1985.

[12] Dickinson M., Thornton A.C., Graves S. Technology Portfolio Management: Optimizing Interdependent Porjects over Multiple Time Periods. IEEE Transaction on Engineering Management (Vol. 48, 2001), 518-527.

[13] Imai, K., Nonaka, I., Takeuchi, H. Managing the New Product Development Process: How the Japanese Companies Learn an Unleran in Clark, K., b., Hayes, R., H., Lorenz, C. (eds.). The uneasy Alliance. Harvard Businnes School Press, Boston, 1985.

[14] Kaner, C., Falk, J. Nguyen, H.Q., Testing Computer Software, Wiley & Sons, 1999.

[15] Kleinrock L Queueing Systems, Vol I: Theory, John Wiley & Sons 1975

[16] Krishnan, V., Ulrich, K., T., Product Development Decisions :A Review of the Literature Management Science, (Vol. 47, 2001), 1-21

[17] Kruchten, P. The Rational Unified Process: An Introduction. Addison-Wesley, 1998.

[18] Kulkarni, V., G., Adlakha, V.,G. Markov an Morkov-Regenerative PERT Networks *Oper. Res. (1986)* Vol. 34, 769-781.

[19] Lavenberg S.S. Computer Performance Modeling Handbook (New York, 1983), Academic Press.

[20] Loch, C. H.: Operations Management and Reengineering, European Management Journal (Vol.16, 1998), 306 – 317.

[21] Petriu D.C., Shen H. Applying the UML Performance Profile:Graph Grammar-based derivation of LQN models from UML specification. Proceedings of Performance TOOLS 2002, London, England, April 14-17 2002, LNCS 2324, Springer Verlag.

[22] Pooley R., Software Engineering and Performance: A Roadmap, (in Finkelstein A. ed.) The Future of Software Engineering, 22nd ICSE.

[23] Putnam L.H., Mayers W. Mesaures for Exellence: Reliable Software on Time, within Budget, (Englewood Cliffs, New Jersey, 1992) Yourdon Press Computing Series.

[24] Ramaswamy R. How to staff businnes critical maintanace projects. IEEE Software (Vol. 7, 2000), 90-95.

[25] Rumbaugh J., Jacobson I., Booch J. The Unified Modeling Language Reference Manual (1999) Addison Wesley.

[26] Selic B. Response to the OMG RFP for Schedulability, Performance and Time. OMG document Ad/2001-06-14.

[27] Smith, C.U. Performance Engineering of Software Systems. Addison-Wesley, Reading, (MA, 1990).

[28] Smith, C.U. and L. Williams. Performance Solutions, Addison-Wesley, 2001.

[29] Smith, R., The Historical Roots of Concurrent Engineering Fundamentals. IEEE Transaction on Engineering Management (Vol. 43, 1997), 67-78.

[30] Takeuchi, H., Nonaka, I., The New Product Development Game. Harvard Business Review (Vol. 64, 1986), 137-146.

[31] The Unified Modeling Language, Advancing the standard, Proceedings of Third International Conference (A. Evans, S. Kent, B. Selic, Eds.), York, UK, October 2000, LNCS 1939, Springer Verlag.

[32] The Unified Modeling Language, Modeling Languages, Concepts and Tools, Proceedings of Fourth International Conference (M. Gogolla, C. Kobryn Eds.), Toronto, Canada, October 2001, LNCS 2185, Springer Verlag.

[33] The Unified Modeling Language, Modeling Engineering, Concepts and Tools, Proceedings of Fifth International Conference (J.-M. Jézéquel, H. Hussmann, S. Cook Eds.), Dresden, Germany, October 2002, LNCS 2460, Springer Verlag.

[34] Wosp2000, Proceedings of Second International Workshop on Software and Performance, Ottawa, Canada, September 2000, ACM press.

[35] Wosp2002, Proceedings of Third International Workshop on Software and Performance, Roma, Italy, July 2002, ACM press.

[36] UML™ Profile for Schedulability, Performance, and Time Specification. On line at htttp://www.omg.org/cgi-bin/doc?ptc/02-03-02.pdf

[37] UML Documentation version 1.4 Web Site. On-line at http://www.rational.com/uml/resources/documentation/

[38] Weiss, G. Stochastic Bounds on Distribution of Optimal Value Function with Application to PERT, Network Flows and Reliability *Oper. Res.* (Vol. 36, 1986), 595-605.

[39] http://www.microsoft.com/office/project/

[40] http://www.iil.com/brochures/kerzner.htm

[41] http://www.gentleware.com

10. Appendix A

With reference to step 4 in Section 4.2, in this appendix we explain the process applied for automatically deriving the EG and the EQNM from the SD and DD designed by the project manager.

10.1 Analysis of the SD

First step: for each SD interaction a tuple (l, A1, A2, *PApar*) is identified where l is the label of the SD interaction arrow, A1 is the name of the SD axis where the arrow starts, A2 is the name of the SD axis where the arrow ends and *PApar* represents the performance annotation of <<PAstep>>.

Step two: The SD, is now translated into a high level EG (called meta-EG). Each node in the EG identifies an interaction, and corresponds to the set of operations performed in relation to that interaction. Every node in the meta-EG is labelled with the tuple (l, A1, A2, *PApar*) that characterizes the translated interaction. Figure 7 illustrates the skeleton of the algorithm used for the EG generation with proper labels for a very simple SD.

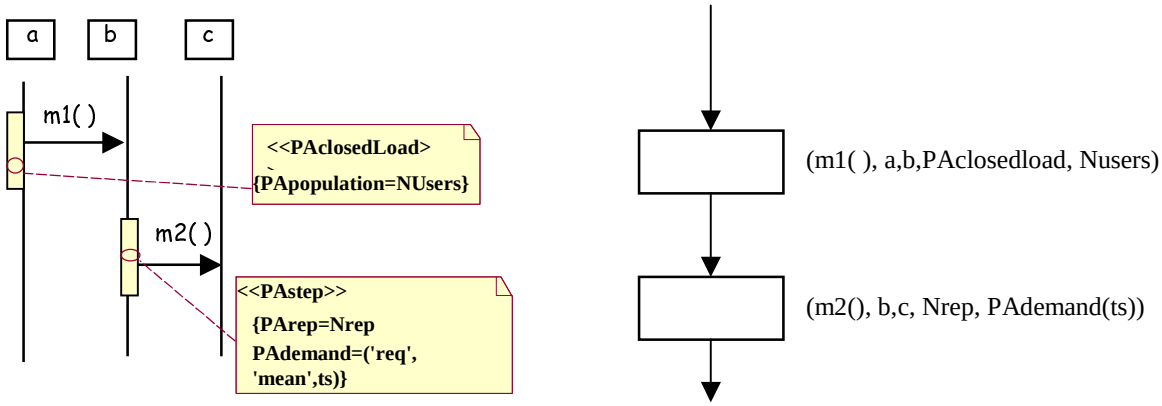

Figure 7: Labeled Meta-EG generation

Depending on the level of detail the manager adopts, the SD could be too complex to allow the generation of a unique comprehensible software model (EG). In this case before generating the EG it could be convenient to check if there are parts (operations in the SD) that can be grouped together. For example, we could aggregate a set of operations that is repeated many times in the SD, or that belongs to a same process phase. In this case a high level EG is generated and expanded, when necessary, into sub-

EG. The Execution Graph, called meta-EG, obtained in this step includes only five types of nodes: basic, branching, cycle, fork and join [27][28]. Each basic node is labeled with the tuple (l, A1, A2, *PApar*) identifying an interaction, and corresponds to the set of operations that are carried out by the component A1 before interacting with A2 through (l, A1, A2, *PApar*). This set of operations is translated into an EG node and possibly connected to another node by a pending arrow. In case of multiple interactions, a fork node is placed before considering the sequence of the outgoing interactions. The multiplicity of the fork node determines the number of pending arrows associated to it and is obviously equal to the cardinality of the multiple interactions (i.e., the number of different threads originated by this interaction). Figures 3 and 4 show the high level EG obtained from the SD in figure 1 and a sub-EG relative to the block problem analysis.

10.2 Analysis of the Deployment Diagram

The usage of the information contained in the Deployment Diagram is twofold. On one hand, the tailoring of the meta-EG to the specific platform can be performed, thus obtaining an EG-instance; on the other hand an Extended Queueing Network Model (EQNM) can be obtained representing the hardware platform hosting the software system.

We start describing the stepwise process adopted for deriving the EG-instance. In our case the nodes of the DD does not represent the hardware resources, but the different teams. The components inside a node represent the tasks that the team must perform (obviously, a team can be composed by one or more people). Project phases can be carried on with the collaboration of components living inside different nodes of the DD. Specifically, the names of the interacting components within the meta-EG block labels are substituted with the names of the team that accomplishes the operation and the values of the relative *PAattribute*. Furthermore, when in the label the names of the interacting components are different, an overhead delay due to coordination among project teams (e.g., team meeting) is added to the performance model. In such a way the node label in the EG-instance corresponds to the *demand vector*, which specifies for each team the work-demand relative to the modeled operation. Figure 4 shows an example of an EG-instance, including the demand vector specification.

The EQNM topology can be derived in a straightforward way from the information collected in the DD. As already stated, in our case the service centers model the project teams involved in the software processes, so the number of service centers in the network correspond to the number of teams. The connections between different service centers are derived from the communications represented in the

DD. The values of the attributes associated to the different node of the DD are used to better specify the characteristics of the EQNM service centers. Successively by using well-known performance techniques [10] [27][28], the obtained EG-instance is combined with the EQNM to achieve the complete definition of the queueing model, as in the traditional SPE approach. The obtained model is then solved by use of classical solution technique and tools [19][28] to obtain the performance indices of interest**.**