Implementation of the Software Performance Engineering Development Process
Salvatore Distefano, Antonio Puliafito, Marco Scarpa
Department of Mathematics, University of Messina, Contrada di Dio, S. Agata, 98166 Messina, Italy
Email: {sdistefano,apuliafito,mscarpa}@unime.it

I. INTRODUCTION

Performance is an important but often overlooked aspect of the software design. Indeed, the consideration on performance issues is in many cases left until late in the software development process (SDP), when problems have already manifested themselves at system test or within the deployed system. The identification of possible bugs or unsatisfactory performance in the design phase allows to contain the costs, also permitting to compare different alternatives. This kind of approach implements the so called software performance engineering (SPE) [1], which is a systematic, quantitative technique to construct software systems that meet performance objectives.

In our previous work [2], [3], we elaborated a technique to carry out the evaluation of the required performance parameters of a software architecture, using the UML [4] for modeling and the OMG UML Profile for Schedulability, Performance and Time Specification (SPT) [5] to specify performance requirements into a UML model therefore mapped into a performance model. More specifically, starting from the UML model, all the software architecture specifications are translated into an intermediate model, the Performance Context Model (PCM), that is subsequently mapped into the corresponding performance model. The results obtained from the analysis can be fed back into the original model for further refinements (re-engineering). Such technique has been implemented into the ArgoPerformance tool [6], a performance plug-in integrated in the ArgoUML CASE tool [7].

The automation process puts in evidence the necessity to formalize the description of a software architecture into the UML domain. A modeler that wants to evaluate the design or the implementation of a software architecture, has to represent it through a model compliant to the software performance engineering tool used, mapping the original model into a performance model thus analyzed. To implement and automate such mapping is therefore necessary to specify representation elements, semantics and rules to univocally and unambiguously represent the software architecture.

In this perspective, the main contribution of this paper is the implementation of a software development process which takes into account performance specifications and requirements: the software performance engineering development process (SPEDP). SPEDP includes and synthesizes both the aim of modeling and developing a generic software architecture, and the aim of investigating the performance of the overall (hardware/software) elaboration system. It can be applied both in early phases, as a software performance engineering technique, and in test phases, as a common software performance evaluation technique. SPEDP fixes steps, rules and guidelines to follow in order to achieve the desired results in the software development, satisfying the performance requirements.

The main advantage of specifying a software development process including performance evaluation is to provide an algorithm, a process for implementing high-quality/performance-guaranteed software. Such a process can be automated into specific tools that, having in input the software design and the performance requirements, can automatically provide as output the final software architecture satisfying the requirements.

The reminder of the paper is organized as follows: in section II an overview of background concepts is provided; then, section III specifies both the syntax and the semantics of SPEDP. In section IV, the SPEDP technique is applied to the modeling of a web video application, and then, in section V, the characterization of SPEDP to the UML/PCM-SPT domain is specified. Finally, in section VI, we present the results of the analysis performed by ArgoPerformance, and, in section VII, we provide some conclusive remarks.
II. BACKGROUND AND PRELIMINARIES

Performance estimation is a crucial aspect of software development. Traditionally software performance are evaluated by specific benchmarks applying the “fix-it-later” principle. In the classical approach, system and software specification, implementation, behavioral and performance validation are disjoint activities, resulting in one model/formalism for each purpose. This may lead to inconsistencies and furthermore is a waste of manpower and resources. Therefore it is desirable to develop a technique that integrates specification, implementation, behavioral and performance validation altogether.

This approach has been identified in the software performance engineering (SPE) [1]. The software performance engineering process begins early in the software life cycle and uses quantitative methods to identify satisfactory designs and to eliminate those that are likely to have unacceptable performance before developers invest significant time in implementation. Software performance evaluation continues through the detailed design, coding and testing phases to predict and manage the performance. SPE begins early in the software life cycle and uses quantitative methods to identify satisfactory designs and to eliminate those that are likely to have unacceptable performance before developers invest significant time in implementation. Software performance evaluation continues through the detailed design, coding and testing phases to predict and manage the performance of the evolving software as well as monitor and report actual performance versus specifications and predictions.

The specific literature offers a lot of works proposing and implementing software performance engineering techniques. Petriu and Woodside, in [8], define a performance meta-model, named Core Scenario Model (CSM), used as an intermediate model. The software architecture behavior and the related performance specifications are detailed through UML behavioral diagrams (Activity Diagrams and/or Sequence Diagrams), annotated with SPT tags and stereotypes. The CSM is based on the SPT profile and can be mapped into several kinds of performance models (simulative models, Petri nets, queueing networks, etc).

Marzolla and Balsamo in [9] propose a software performance engineering technique which uses a representation notation composed by a set of annotated diagrams (Use Case, Activity and Deployment). The software architecture behavioral description is implemented by higher level Use Case Diagrams, detailed by Activity Diagrams. The nodes of the Deployment Diagram correspond to software architecture elaboration system devices. They propose a “UML Performance Simulator” which transforms the UML software architecture model into a discrete-event simulation model. Recently, they extended their technique to the analytic performance domain by defining a specific mapping into LQN [10].

In [11] the authors establish a correspondence between UML models and labeled generalized stochastic Petri nets (LGSPN) in two steps: first each UML State Machine Diagram, composing the software architecture behavioral UML description, is independently converted into the corresponding LGSPN [11]; then, the PNs thus obtained are joined according to the information reported in the UML Sequence and Use Case Diagrams that also contain the performance specifications. An extension of such technique is proposed in [12], where UML State Machines and Activity Diagrams are used.

Another software performance engineering technique using reward models is proposed in [13], in which the authors develop an intermediate modeling language for component-based systems’ performability evaluation called KLAPER. According to such technique, software components are represented by Component Diagrams. The behaviors of the components’ services are detailed by Activity Diagrams and/or State Machine Diagrams enriched by UML-SPT [5] and UML-QOS [14] profiles annotations. Finally, a Deployment Diagram describes the deployment of the components on the elaboration infrastructure. This UML model is translated into the KLAPER intermediate model, then into a semi-Markov reward model and therefore analyzed.

Smith et al. in [15] and Gu and Petriu in [16] use or specify XML schema meta-models or Document Type Definitions for representing performance specifications into UML models, the Performance Model Interchange Format (PMIF) and the eXtensible Stylesheet Language for Transformations (XSLT), respectively. Smith et al. in [17] make use of queueing networks for solving their UML annotated (Sequence Diagnostics) software model previously translated into a PMIF–XML intermediate model. A similar approach is specified in [18] where Gu and Petriu use an XML algebra-based algorithm to transform UML annotated models in XSLT format, and then to the corresponding layered queueing network.

Some other interesting works related to the software performance engineering topic, updated to 2003, are summarized in [19].

In [2], [3], [6] we provided a SPE technique for evaluating the performance of a software architecture modeled by UML diagrams, based on the PCM intermediate model, thus splitting the mapping from UML to performance model into two phases. Such technique has been automated into the ArgoPerformance tool [6], [20], a performance plug-in integrated in the ArgoUML CASE tool [7] implementing a solution based on non-Markovian stochastic Petri nets (NMSPN). As introduced above, in order to automate such mappings, the transformation rules allowing to translate the original UML model into the destination (intermediate and/or performance) model have to clearly and unambiguously specified and formalized. Thus, it is necessary to formalize the syntax and the semantics of the representation notation, so that the source models (UML and PCM for ArgoPerformance) result unambiguous to the tool that automatically maps them into the target model (PCM and NMSPN respectively in the ArgoPerformance case). In all the above referred works remain ambiguous or unspecified how a model has to be built, which diagrams are used and what to model/specify by them. A modeler knows how to specify the performance annotations but he/she does not know how to represent the software architecture to be evaluated in order to build a consistent, (automatically) analyzable XSLT, PMIF, CSM, ...-compliant model.

In this work, instead of only providing the rep-
representation guidelines to be used for implementing ArgoPerformance-compliant models, we choose to specify a whole software development process taking into account performance requirements and specifications, the software performance engineering development process. Particular attention is focused on the modeling phase, formalizing rules and guidelines to make unambiguous performance annotated software models. In this way it will be possible to evaluate the software architecture performance by exploiting our SPE technique and its automated implementation. Through SPEDP we specify generic representation rules and guidelines, that are then characterized into the UML-SPT domain.

The idea of specifying a software development process integrating performance issues is an enhancement, a step forward, the completion of our original UML-SPT-PCM SPE technique. The main advantage of specifying a performance driven software development process lies in the possibility of providing a whole process that, having in input the software architecture model and the performance requirements, brings to the implementation of a software architecture that meets the requirements. In other words, the SPE approach only considers the problem of obtaining performance indexes from a software model/design, while a performance driven software development process and in the specific SPEDP, tries to also specify rules and guidelines to follow thoroughly the SDP in order to satisfy specific performance requirements.

III. THE SOFTWARE PERFORMANCE ENGINEERING DEVELOPMENT PROCESS

This section details the SPEDP, providing an high level description, in subsection III-A, and then specifying (subsection III-B) its syntax and semantics by exploiting the MDE meta-modeling technique.

A. High Level Point of View

Before entering into the details of the semantics specification, it is useful to describe the approach from an higher level point of view.

A SPEDP software architecture model is composed of two parts: the static part, in which the static organization of the software, its data structures and the relationships among such data are represented in an object-oriented way; and the dynamic part, which describes the software behavior and its interactions from both the internal and the external points of view. With regards the static part description, since we have no specific and/or particular requirements, it does not make sense to redefine an existing formalism. Thus we decided to adopt the UML specification and more specifically the Class Diagrams. Class Diagrams allow to represent any type of classes, attributes and relationships among classes and attributes (association, aggregation, composition, generalization, realization, dependency, multiplicity, etc.). In this way SPEDP inherits the character of generality of UML in modeling the static organization of the software.

Concerning the dynamic part, SPEDP can be considered as a characterization of UML, providing a stronger semantics than UML. SPEDP defines a specific process that, if followed, drives in developing a performance model. So, in the following we only consider the dynamic part of the software architecture, describing how it is expressed by SPEDP.

The first step a user should do to describe a software architecture following SPEDP is to identify and separate independent processes, tasks or critical parts, decomposing the global project into several logical blocks. The blocks, whose behaviors are considered of interest in terms of performance, are selected to be further investigated: they must be detailed defining workloads, resources allocation requirements and behaviors. All such information are translated into a three logical views representation, pictorially depicted in Fig. 1.

The Global view is devoted to represent the general aspects of the system. The Specification view characterizes the independent parts of the overall software architecture selected for a performance analysis, coming into the description of the overall logic organization (requirements, workloads, resource deployment). The Detailed view defines the referred block(s) from a behavioral point of view.

The complexity of the software architecture is reflected in the corresponding SPEDP model in terms of a Global view levels’ hierarchy. Referring to Fig. 1, at least one level must compose such hierarchy: the lower level GV Organization Diagram (GVOD:x,y,...,z), which Scenarios S1, ..., Sn must be alternative (only one of them can be performed at time) or absolutely not overlapped (they are not interacting, totally independent, different tasks, resources and workloads). In order to investigate the performance of such Scenarios, each of them must be adequately described by the Specification and the Detailed views. The Specification view Organization Diagram Sc-
narios ($S_i$) describe interacting parts of the SA. They can share workloads and resources. Their behaviors are detailed by the corresponding Behavior Diagrams.

**B. The SPEDP Specification**

In this subsection we provide the meta-modeling specification of the SPEDP representation notation, by adopting the MDA-MOF approach.

The packages organization of SPEDP describing its logical organization is reported in Fig. 2. The SPEDP Core package specifies the relationships between the diagrams and the views according to subsection III-A. The Statics package contains elements to characterize the SA architecture, such as resources, functionalities, services and workloads. The resource deployment of the SA is modeled through the elements contained in the Resource package, while the SA functionality features are described by the Organization package. The SA behavior is modeled by the elements of the Behavior package. Finally, the Performance Requirements package includes elements characterizing the performance domain, such as probability density functions (PDF) and scheduling policies.

In order to describe the approach adopted in the SPEDP specification we only provide in the following the meta-model formalization of the SPEDP Core sub-package. Details on the specification of the other sub-packages can be found in [21].

**SPEDP Core** is the most fundamental sub-package composing the SPEDP package. It defines the basic constructs needed for the development of SPEDP compliant models.

![Figure 2: The SPEDP Packages Organization.](image)

![Figure 3: The SPEDP Core Abstract Syntax meta-model.](image)

**a) Abstract Syntax:** The SPEDP Core meta-model reported in Fig. 3 describes how a SPEDP model has to be built. A model is represented by the Model class on the top of the hierarchy, associated to the GVOrganizationDiagram class by the composition relationship with multiplicity of 1. The GVOrganizationDiagram class is related to itself by parent/child associations with multiplicity of [0..1] and *[ ]*, respectively, in order to implement the SPEDP global view hierarchy. The GVOrganizationDiagram class is associated to both SVOrganizationDiagram and ResourceDiagram classes implementing the SPEDP specification view by the composition relationship, both with the same *( [ ] )* multiplicity for the component classes. The association between SVOrganizationDiagram and ResourceDiagram is two-way, the ods association end has multiplicity *[ ] *, while the other end has multiplicity 1. The SVOrganizationDiagram and GVOrganizationDiagram specify the OrganizationDiagram class. Finally, the composition relationship links the SVOrganizationDiagram and the BehavioralDiagram implementing the SPEDP detailed view, with *[ ] * multiplicity at the bd end.

**b) Well-formedness Rules:** The well-formedness rules (wfrs) regarding the SPEDP Core package are reported in the following, defined and formalized through OCL constrains.

```
context GVOrganizationDiagram
inv base: self.size() ≥ 1
self.svod --- forAll(sv: SVOrganizationDiagram | sv.bd.size() ≥ 1)
```

**a) wfr base**

```
context GVOrganizationDiagram
inv gvsucc:
if self.child -> forAll(sv: SVOrganizationDiagram | sv.size() == 0)
then
  self.svod --- forAll(sv: SVOrganizationDiagram | sv.size() > 0) and
  self.rd --- forAll(res: ResourceDiagram | res.size() > 0)
else
  self.svod --- forAll(sv: SVOrganizationDiagram | sv.size() == 0) and
  self.rd --- forAll(res: ResourceDiagram | res.size() == 0)
```

**b) gvsucc wfr**

Figure 4: SPEDP Core package wfrs.

i) The **base** rule, reported in Fig. 4a, specifies that the Model must be composed of at least one GVOrganizationDiagram element. Moreover, all the SVOrganizationDiagram composing a GVOrganizationDiagram, must be detailed by a set of BehavioralDiagram elements.

ii) The **gvsucc** wfr of Fig. 4b specifies that the lower level GVOrganizationDiagrams, the leaves of the GVOrganizationDiagrams hierarchy of Fig. 1, characterized by attribute child == null, must be composed of GVOrganizationDiagram and ResourceDiagram elements.

**c) Detailed Semantics:** The SPEDP Model represents the whole SA to be modeled. A Model is composed of GVOrganizationDiagram instances. The GVOrganizationDiagrams identify the main independent/alternative scenarios of the SA, specifying the interactions between the SA and the external environment. According to the gvsucc wfr of Fig. 4b, the scenarios contained into the leaves of the GVOrganizationDiagrams parent/child hierarchy are selected to be investigated in terms of perfor-
mance. SVOrganizationDiagram and ResourceDiagram objects implement the Specification view, representing the internal organization and the resource deployment of the selected scenarios, respectively. In the Detailed views, the BehavioralDiagrams implements the Detailed view of the subsystem under exam, characterizing the dynamic behavior of each scenario specified in a SVOrganizationDiagram.

IV. USING SPEDP: A WEB-BASED VIDEO APPLICATION

In this section we describe how the SPEDP is applied to the development of a web-based video application, extending an example presented in [2], [3], also investigated in [9] and originally taken from the OMG SPT specification [5]. The videos, uploaded into the system by the administrator, can be listed by a generic user belonging to the surfer or the client categories and can also be requested and downloaded only by registered users (clients). Below, we present the modeling of such application according to the SPEDP guidelines, by specifying the three views (Global view, Specification view and Detailed view) following the specifications provided in subsection III-A.

A. SPEDP Modeling

The SPEDP model representing the software architecture under exam is composed by a GVOrganizationDiagrams, a SVOrganizationDiagrams and four BehavioralDiagrams implementing the global, detailed and specification views respectively, as shown in the diagram of Fig. 5a representing the overall SPEDP model. The first step of the SPEDP representation algorithm is the identification of the independent parts of the software architecture under exam. The GVOrganizationDiagram of Fig. 5b represents the Global view of the web-based video application.

The VideoManagement scenario specifies the access policy regulating the requests incoming to the web-based video application. At most six requests are accepted by the application (access.capacity=6) and three of them can be managed concurrently (access.capacity=3). The VideoManagement scenario is associated by the Ass object to the User Stakeholder representing the generic interactions with the environment.

The next step is to detail the GVOrganizationDiagram’s VideoManagement scenario by specifying its specification view through the SVOrganizationDiagram and the ResourceDiagram of Fig.s 6a and 6b, respectively.

The SVOrganizationDiagram of Fig. 6a characterizes the global view User Stakeholder with three WorkLoads: Surfer, Admin and Client. The Surfer object is an open workload (population=1), while there is only one administrator (Admin.population=1) and ten registered users (Client.population=10). Each WorkLoad, characterized by a PDF representing the interarrival time, is probabilistically associated to the corresponding Scenario by the Association instance. Three Scenarios are identified in the SVOrganizationDiagram: the Browsing, the RequestVideo and the UploadVideo, to be further specified by the SPEDP detailed view.

The ResourceDiagram of Fig. 6b describes the architecture of the computing system elaborating the application. The ClientWorkstation, WebServerNode, VideoServerNode and Internet objects are ActiveResource instances, associated to each other by Associations. The ClientWorkstation, the WebServerNode and the VideoServerNode are managed by FIFO policies (schdPolicy=FIFO), while the Internet object is managed by a preemptive resume policy (schdPolicy=PreemptiveResume).

The last step of the SPEDP modeling algorithm is to investigate the behavior of the application under exam by specifying its detailed view. Each scenario of the specification view is detailed by a BehavioralDiagram. Fig. 7 shows the dynamic behavior of the Browsing scenario by specifying two nested BehavioralDiagrams: the main flow is depicted in Fig. 7a, while Fig. 7b reports the ProcRequest (sub-)BehavioralDiagram. All the Process objects are associated to the corresponding Resources (ClientWorkStation and WebServerNode in the specific).

The execution time spent in such Resources by the BehaviorProcess tasks (RequestPage, SendContent, ShowPage, GetCookie, LoadProperties, CreateDynamicPage) are specified into the demandPDF attributes. ProcessingRequest is a particular BehaviorProcess without attributes since it models the (sub-)BehavioralDiagram connection. The br1 Branch object of Fig. 7b models a selection between two BehaviorProcesses, whose condition’s verification is probabilistically quantified by the prob attribute of the following Transitions. Similarly, Fig. 8 describes the dynamic behaviors of the RequestVideo and UploadVideo scenarios. Fig. 8a highlights the parallel computing, represented by the Fork and Join objects.

V. IMPLEMENTATION OF SPEDP ON THE UML–PCM DOMAIN

In this section, we describe the implementation of SPEDP into the UML–PCM domain. As introduced in subsection III-A, the SPEDP static part is fully specified by the UML standard through the class diagrams.
syntax and semantics. In this section we focus on the dynamic/behavioral part of SPEDP specified in section III. More specifically, we specify how to map SPEDP elements into corresponding UML–PCM elements by following the modeling of the example discussed in the previous section. The UML–PCM domain derives from the UML OMG specifications [4], [22] and also from the PCM-SPT extending the SPT OMG profile [5]. Details on PCM–SPT can be found in [21].

A. Core & Performance

<table>
<thead>
<tr>
<th>SPEDP</th>
<th>UML</th>
<th>PCM-SPT Sp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Model</td>
<td>Model</td>
</tr>
<tr>
<td>OrganizationDiagram</td>
<td>Use Case Diagram</td>
<td>Use Case Diagram</td>
</tr>
<tr>
<td>ResourceDiagram</td>
<td>Deployment Diagram</td>
<td>Deployment Diagram</td>
</tr>
<tr>
<td>BehavioralDiagram</td>
<td>Interaction/Activity/StateChart Diagram</td>
<td>Interaction/Activity/StateChart Diagram</td>
</tr>
<tr>
<td>PDF</td>
<td>PAperfValue, RTarrivalPattern</td>
<td>PAresource</td>
</tr>
<tr>
<td>Policy</td>
<td>PAresource</td>
<td>PAresource</td>
</tr>
</tbody>
</table>

TABLE I.: Correspondences among SPEDP Core/Performance packages and UML–PCM-SPT elements.

Table I specifies the correspondences between the SPEDP Core and Performance packages’ elements and the UML–PCM elements. This is the most fundamental among the tables, referred in all the following subsections. From this table we can argue that a UML–PCM Model implementing the SPEDP technique is composed by just Use Case Diagrams, Deployment Diagrams, Activity Diagrams, StateChart Diagrams and Interaction Diagrams (Sequence and/or Collaboration Diagrams).

B. OrganizationDiagram

<table>
<thead>
<tr>
<th>SPEDP</th>
<th>UML</th>
<th>PCM-SPT Sp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario</td>
<td>Use Case</td>
<td>PAcontext</td>
</tr>
<tr>
<td>Stakeholder</td>
<td>Actor</td>
<td>PAcontext</td>
</tr>
<tr>
<td>Workload</td>
<td>Actor</td>
<td>PAcontext</td>
</tr>
<tr>
<td>Association</td>
<td>Association</td>
<td>PAcontext</td>
</tr>
<tr>
<td>Relationship</td>
<td>Extend</td>
<td>PAcontext</td>
</tr>
</tbody>
</table>

TABLE II.: Correspondences among SPEDP Organization package and UML–PCM-SPT elements.

Since, as specified in Table I, the SPEDP OrganizationDiagram corresponds to a UML Use Case Diagram,
the elements of the former have a corresponding element in the latter, as reported in Table II.

**SPEDP Scenarios** are mapped by UML Use Cases, annotated by the PAcontext PCM-SPT stereotype, if they belong to a leaf of the GVOrganizationDiagram tree, since they represent independent SA subsystems to further detail and investigate in terms of performance. Stakeholders correspond to Actors, modeling Workloads characterized by population=1 (open) or population>0 (closed) if they are annotated by the PAopenLoad or PAclosedLoad stereotypes, respectively. The **SPEDP Association** is mapped by UML Association characterized by the PAstep annotation in order to model the probabilistic association.

Fig. 9a shows the web-video application UML–PCM global view Use Case Diagram corresponding to the **SPEDP GVOrganizationDiagram** of Fig. 5b. It is obtained by applying the mapping rules specified in Tables I and II. The names used in the **SPEDP GVOrganizationDiagram** corresponds to the ones specified in the Use Case Diagram. So, they both specify a VideoManagement **SPEDP Scenario**/UML–PCM Use Case representing the video management functionalities, and a User Stakeholder/Actor to model the generic interactions, as reported in Fig. 9a. The PAcontext PCM-SPT annotation associated to the VideoManagement **Use Case** specifies its access policy, represented by the specific attributes of the **SPEDP Scenario**.

In the same way the **Use Case Diagram** of Fig. 9b models what specified by the SVOrganizationDiagram of Fig. 6a in the **SPEDP** domain. Thus, Surfer, Admin and Client Actors annotated by PAworkload stereotypes correspond to the Surfer, Admin and Client Workload objects of the SVOrganizationDiagram of Fig.6a, respectively. The Admin and Client are closed workload (population ≥ 1), while the Surfer is an open workload (population = −1). Consequently, the Admin and Client Actors are PAClosedLoad stereotyped, while the Surfer is associated to the PAopenLoad stereotype. The PApopulation and PAextDelayActor tags in Fig. 9b, are deduced by the corresponding population and pdfWork **SPEDP Workload** attributes. The probability of the associations between Actor and Use Case is modeled by stereotyping the UML Association with PCM-SPT PAsteps, specifying in the PAprob tag such probability, in the specific case drawn from the corresponding prob Association values in Fig.6a.

**C. Resource**

Table III reports the mapping among the **SPEDP ResourceDiagram**’s elements to the UML/PCM-SPT stereotyped Deployment Diagram elements.

By applying it to the ResourceDiagram of Fig. 6b describing the specification view computing system architecture of the web-based video application, the Deployment Diagram of Fig. 10 is obtained. The ClientWorkstation, ServerNode, VideoServerNode
and Internet ActiveResources are mapped into ClientWorkStation, WebServerNode, VideoServerNode and Internet UML Nodes, respectively. Such Nodes are all associated to PAhost stereotypes, mapping the schdPolicy and power Resource attributes with the corresponding PAhost and PArate tags.

D. Behavior

Table IV reports the mapping from SPEDP BehavioralDiagram to UML 2 [22] PCM-SPT annotated Activity Diagram (AD), StateChart Diagram (SCD) and Interaction Diagram (ID) (Sequence, Collaboration and Communication Diagrams), characterized in the corresponding columns. The last column of such table reports the PCM-SPT stereotypes associated to the UML diagrams’ elements identified by the acronym (AD, SCD, ID). UML PAstep stereotyped Activity Edge of Activity Diagrams and Transitions of StateChart Diagrams map SPEDP probabilistic Transitions outgoing from Branches, while in Interaction Diagrams these latter correspond to CombinedFragments with Interaction-OperatorKind=’‘opt’’ objects, also annotated by PAstep stereotypes. In the specific case where the BehavioralDiagram contains a Merge followed by the matching Branch, this construct represents a loop, and it is mapped into UML Interaction Diagrams by specific CombinedFragments (Interaction-OperatorKind=’‘loop’’).

In the same way, a Fork-Join statement has corresponding elements in both Activity Diagrams (ForkNode, JoinNode) and StateChart Diagrams (PseudoState -Kind=‘‘fork’’ and -Kind=‘‘join’’), while the whole statement corresponds to a specific CombinedFragment with Interaction-OperatorKind=’‘par’’ in UML 2 Interaction Diagrams. The SPEDP Resource objects are mapped into Partitions swim-lanes of Activity Diagrams into Regions of StateChart Diagrams and into Interaction Diagrams’ Lifelines, characterized by the corresponding Deployment Diagram Components/Nodes’ names.

Figure 10: Specification view resources of the web-based video application in UML–PCM domain

Figure 11: The Detailed view Browsing scenario of the web-based video application in the UML–PCM domain. One of the main benefit of the UML/PCM-SPT characterization of SPEDP is the possibility to represent the behavior of the SA under exam by exploiting several different diagrams, also nesting diagrams and formalisms into hierarchical models. As an example, in Fig. 11 we represent the dynamic behavior of the Browsing scenario by using UML Activity, Statechart and Sequence Diagrams mapping the SPEDP BehavioralDiagram of Fig. 7, as stated by Ta-
Table IV: Correspondences among SPEDP Behavior package and UML/PCM-SPT elements.

<table>
<thead>
<tr>
<th>Process</th>
<th>Activity AD</th>
<th>UML AD</th>
<th>UML SCD</th>
<th>UML ID</th>
<th>PCM-SPT Sp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Activity</td>
<td>Message</td>
<td>PAstep</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Transition</td>
<td>MessageEnd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>OperatorKind</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CombinedFragment</td>
<td>Interaction-OperatorKind</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fork</td>
<td>Merge</td>
<td>Runtime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Join</td>
<td>Merge</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td>Initial</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End</td>
<td>FinalNode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource</td>
<td>ActivityPartition</td>
<td>Region</td>
<td>Lifeline</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table IV: Correspondences among SPEDP Behavior package and UML/PCM-SPT elements.

ble I. The diagram of Fig. 11a is composed by two hierarchical Activity Diagrams levels: the Browsing one and the ProcRequest sub-diagram. In order to highlight the versatility of the approach and the equivalence, in it, of Activity, Statechart and Sequence Diagrams, we represent the ProcRequest sub-diagram by the SubActivity diagram of Fig. 11a (grey part), the StateChart Diagram of Fig. 11b and the Sequence Diagram reported in Fig. 11c. The link from the higher level diagram to the sub-diagram is implemented by annotating the Processing Request ActionState of Fig. 11a by a PAstep specifying in the PAextOp the name of the sub-diagram and the mean number of times it is invoked (PAextOp=('ProcRequest',1)).

According to Table IV, the SPEDP BehavioralDiagram's BehaviorProcess objects correspond to UML Activity Diagrams' Actions, StateChart Diagrams' States and Sequence Diagrams' Messages. The concept of BehaviorProcess is close to both the Action and the State one, as confirmed by the similarity of the corresponding Browsing models shown in Figs. 7, 11a and 11b, respectively. But it is substantially different from the Message one representing interactions. Thus, in the Sequence Diagram of Fig. 11c, the BehaviorProcesses are represented by Messages exchanged by the ClientWorkStation and WebServerNode Lifelines mapping the corresponding ActiveResources of the BehavioralDiagram of Fig. 7.

In this way the RequestPage, GetCookie, Load-Properties, CreateDynamicPage, SendContent and ShowPage BehaviorProcesses are represented by the corresponding PAstep annotated Actions in the Activity Diagram of Fig. 7, States of the StateChart Diagram of Fig. 11b and Messages of the Sequence Diagram of Fig. 11c. The PAstep's attributes and tags are obtained by the corresponding BehaviorProcess instances' attributes. The brf Branch/Merge objects of Fig. 7b are mapped by a DecisionNode and a MergeNode in the sub-activity of Fig. 11a, and by two PseudoStates (Kind='choice' and Kind='junction', respectively) in the sub-statechart of Fig. 11b, where they are represented by a CombinedFragment (Interaction-OperatorKind='opt') composed by two parts identified by the guards assigned to the LoadProperties and CreateDynamicPage Messages in the sub-sequence diagram of Fig. 11c.

Figs. 12a and 12b show the dynamic behavior of the Request and Upload scenarios by means of UML Activity Diagram. They correspond to the SPEDP BehavioralDiagrams of Figs. 8a and 8b, respectively.

VI. PERFORMANCE EVALUATION THE WEB-BASED VIDEO APPLICATION

The example described thoroughly the paper has been implemented and analyzed by the ArgoPerformance tool [6], [20]. The results obtained are depicted in Fig. 13, that shows the utilization of the resources specified in the Deployment Diagram of Fig. 10 (Client Workstation, WebServer VideoServer and Internet), within 15000 seconds. In order to compare such results, we plot them altogether, using a logarithmic scale for the utilization, in the ordinate axis.

All the trends thus identified reach the steady state after a transient phase. More specifically, since all the users exploit the Client Workstation to access the system, the utilization trend has a maximum value of 86% after 234 seconds, then it slowly tends to the steady state value of 67.10%. On the other hand, since the WebServerNode is involved in the elaboration of the Browsing and the Request scenarios its utilization (steady state utilization: 11.76%) is lower than the Client Workstation one. Anyway the WebServer has faster transient and higher values of utilization than the VideoServerNode ones (steady state utilization: 0.77%). This is due to the low probability to elaborate the scenarios involving the VideoServerNode: Request and Upload. The former is occasionally performed by the client (13% of

© 2010 ACADEMY PUBLISHER
the total cases), and the latter is invoked by the administrator, that has the lowest arrival rates among the workloads. Finally, the Internet is involved in the SendVideo and UploadVideo activities (Fig. 12): their computation implies an utilization (steady state utilization: 48.16%) higher than WebServerNode and VideoServerNode. From such results we can argue that the most critical resource for the system is the Client Workstation. Therefore, in order to improve the performance of the overall SA, it is necessary to enhance and/or replicate the resources from which a user can access to the computing system. Moreover, to improve the system performance further Internet resources (bandwidth) are required as shown in Fig. 13a. On the other hand, since both the VideoServer and the Webserver Nodes show low utilization they can manage higher workloads incoming by increasing Client Workstation and Internet resources.

Fig. 13b, show similar trends for the Client Workstation and the Webserver Node throughput, while lower throughput are experienced for the VideoServer Node and especially for the Internet.

VII. CONCLUSIONS

In previous works we defined an SPE technique based on the intermediate Performance Context Model (PCM) that automates the translation of the project requirements from the design domain to a performance domain by the ArgoPerformance tool. This has highlighted the necessity of formalizing the description of a software architecture. In this paper we address the problem from a wider point of view, defining a software development process which takes into account performance specifications and requirements, an integrated software performance engineering development process (SPEDP). In this way, a whole process starting from the software architecture model and the performance requirements aiming at implementing a software architecture that meets the requirements thoroughly all the software design phases is specified. It is more powerful than both an SPE technique, since it can be applied to existing software architecture as a common performance evaluation methodology, and a software performance evaluation technique, since it allows
evaluation at early, design stage (SPE).

SPEDP has been formalized by using the model-driven engineering meta-modeling approach. In this way, by characterizing the PCM-SPT annotated UML as modeling domain, we specify the methodology we implemented in the ArgoPerformance tool, thus applied to a Web-based video application example.

REFERENCES


Marco Scarpa received his degree in computer engineering in 1994, from University of Catania and the Ph.D. degree in computer science in 2000, from University of Turin. He is currently associate professor in “Performance Evaluation” at the University of Messina. He coordinated the development of the software package WebSPN. His interests include performance and reliability modelling of distributed and real time systems, phase type distributions, distributed systems, and Software Performance Evaluation techniques. He has been involved in several research projects.

Antonio Puliafito is a full professor of computer engineering at the University of Messina, Italy. His interests include parallel and distributed systems, networking, wireless and GRID computing. He was a referee for the European Community for the projects of the fourth, sixth, and seventh Framework Program. He has contributed to the development of the software tools WebSPN, MAP, and ArgoPerformance. He is co-author (with R. Sahner and K. S. Trivedi) of the text entitled “Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package”, edited by Kluwer Academic Publishers.