

# **Information Requirements for Software Performance Engineering**

Lloyd G. Williams<sup>†</sup> and Connie U. Smith<sup>§</sup>

<sup>†</sup>Software Engineering Research, Boulder, Colorado USA  
<sup>§</sup>Performance Engineering Services, Santa Fe, New Mexico USA

Copyright © 1995, Performance Engineering Services and Software Engineering Research

All rights reserved

This material may not be sold, reproduced or distributed without written permission from  
Performance Engineering Services or Software Engineering Research

This material is based upon work supported by the National Science Foundation  
under award number DMI-9361824. Any opinions, findings, and conclusions  
or recommendations expressed in this publication are those of the author(s) and do not  
necessarily reflect the views of the National Science Foundation.



## **1 Introduction**

The design and construction of future software systems will require the integration of software analysis and design methods with Software Performance Engineering (SPE) [Smith, 1990 #485]. Software methods provide rules and guidelines for performing systems analysis and designing software. Software Performance Engineering [Smith, 1990 #480] is a method for constructing software systems that meet performance goals. SPE includes techniques for gathering data, coping with uncertainty, constructing and evaluating performance models, evaluating alternatives, and verifying and validating results. It also includes strategies for the effective use of these techniques.

Software methods and SPE methods evolved independently. As a result, there are several barriers to their integration. The notations used for constructing analysis/design models and SPE models are very different. In addition, the information maintained in the repositories of CASE tools that support software analysis/design methods contain some, but not all, of the data needed by SPE tools. Finally, software analysis/designers are rarely trained in SPE methods and techniques. Typically, software designs are created by one team while their performance is analyzed by another. This artificial division means that there is little opportunity to explore design alternatives based on their performance characteristics.

There is, however, no reason why current software analysis/design methods and SPE cannot be more closely integrated. This integration would allow software designers to explore design alternatives and select a design that provides the best overall combination of understandability, reusability, modifiability and performance. Easy, early evaluation of performance alternatives would also help to eliminate the “fix-it-later” approach in which performance issues are ignored until after the software has been constructed. Performance problems discovered after the software has been implemented are typically addressed by tuning, a process which can introduce errors and ruin a carefully constructed design. In addition, systems that have been tuned rarely perform as well as those that have been designed with performance in mind.

This paper explores the information requirements for Software Performance Engineering and their relationship to the process used and information gathered in contemporary systems analysis and software design methods. We begin by cataloging the information needed to construct and evaluate early life-cycle performance models. This information is then used to construct a Software Performance Engineering meta-model. This SPE meta-model can be used to define the performance information that should be captured during the analysis and design process. Most, or all of this information could be captured by current CASE tools. The meta-model can also serve as the basis for defining an transfer format that allows CASE tools to export performance data to performance modeling tools for evaluation.

## **2 Related Work**

In the research community, there are several notable approaches to integration of CASE and SPE. The CAEDE system, developed at Carleton University, is a comprehensive

CASE tool that includes analytic performance models of reactive systems under design [Woodside, 1989 #622]. The performance models are based on Rendezvous nets and thus permit the analysis of Ada Rendezvous. They are, however, adapted to problems encountered in telecommunication system design, thus the performance metrics they produce focus on throughput measures rather than response time compliance.

Likewise, the Microelectronics and Computer Technology Corporation (MCC) has incorporated performance analysis capabilities into Verdi, their visual tool for designing distributed systems [Shen, 1990 #779]. Among other functions, it depicts a color “animation” of system behavior (by highlighting line drawings of the design) and reports times (mean, variance, and distribution) between start and end markers.

Martin proposed data-action graphs as a representation that facilitates the integration of various software design notations and a performance simulator [Martin, 1988 #1246]. Rolia extends the SPE models and methods to address systems of cooperating processes in a distributed and multicomputer environment with specific applications to Ada [Rolia, 1992 #621]. Woodside proposes stochastic rendezvous networks to evaluate performance of Ada systems [Woodside, 1989 #622]; and Woodside and co-workers incorporate the analysis techniques into a software engineering tool [Woodside, 1991 #623], [Buhr, 1989 #96]. Baldassarri and coworkers integrate a petri net design notation into a CASE tool that provides performance results [Baldassarri, 1989 #613]. Lor and Berry automatically generate SARA models from a requirements specification language using a knowledge-based Design Assistant [Lor, 1991 #1247]. Valderruten and colleagues use performance annotations with LOTOS process algebra specifications and apply a set of rules to produce a queueing network model in [Valderruten, 1992 #1251].

Opdahl and Sølvsberg augment information system models and performance models with extended specifications [Opdahl, 1992 #1248]. Opdahl describes an SPE tool interfaced with the PPP CASE tool and the IMSE environment for performance modeling – both are part of the European Esprit research initiative. The IMSE is a comprehensive environment to support performance modeling [Opdahl, 1992 #1249]. It interfaces a number of different system performance modeling tools, but has no automatic translation between them. It provides support for model creation and model experimentation. Opdahl's tool supports data acquisition from the PPP CASE tool and supports the initial model creation SPE step. It is an important step for the Esprit project which seeks to integrate a complementary set of tools into a comprehensive development environment.

These approaches focus on an integrated tool set that has both software design and performance analysis features. This paper formally defines the information requirements for SPE independent of the tool used for performance analysis or software design. This definition, the SPE meta-model, serves two purposes. The first is to provide a rigorous definition for the information requirements for early lifecycle Software Performance Engineering. This model is valuable to performance analysts since it is the first rigorous definition of SPE information requirements. It is valuable to CASE tool vendors since it defines the information that they must capture if they are to support SPE functions. The second purpose served by the meta-model is to form the basis for defining an interchange format so that CASE and performance tools can

exchange information. The meta-model currently defines the information that must be exported by a CASE tool to a performance tool.

### 3 Information Requirements for Early Life-Cycle Performance Analysis

Performance analysts need a number of different pieces of information in order to construct and evaluate early life-cycle performance models. These information requirements fall into the following categories [Smith, 1990 #480]:

- performance objectives
- workload specifications
- software plans
- execution environment
- resource requirements
- processing overhead

Workload specifications and software plans are, together, used to construct *performance scenarios* which are the basis for performance models.

The particular information required in each category depends strongly on the problem at hand. In many cases, it may also be possible to express the information in several different ways. Currently, determining what information is required and the most appropriate way of expressing it requires expert judgment. For example, performance goals depend on how the software will be used and whether response time or throughput is most important. Resource requirement specifications depend on the type of system that the software will execute on and the devices most likely to be bottlenecks.

The following sections describe these information requirements in more detail.

#### 3.1 Performance Objectives

Performance objectives specify quantitative criteria for evaluating the performance characteristics of the system under development. These objectives may be expressed in several different ways, including: response time, throughput, or constraints on resource usage. For information systems, response time is typically described from a user perspective, i.e., the number of seconds to respond to a user request. Throughput requirements are specified as a number of transactions to be processed per unit time.

For real time systems, response time is given as the amount of time required to respond to a given external event. Throughput is specified as the number of events to be processed per unit of time. Jaffe and Leveson [Jaffe, 1991 #628] distinguish two components of throughput for real-time systems: *capacity* and *load*. Capacity refers to the number of events of a given type that the system must be able to process in a given amount of time. Load refers to the number of events of (multiple) different types that must be processed in a given amount of time. For each kind of event, Jaffe and Leveson recommend defining a minimum and maximum arrival rate.

Resource constraints describe limits on the amount of services required from key devices in the hardware configuration. These may be expressed as constraints on processor utilization (e.g., percent of CPU capacity), limitations on memory use, and so on.

## **3.2 Performance Scenarios**

Performance scenarios model a particular use of the system and the demands that it makes on the available resources. The model can be used to predict the performance characteristics of the proposed software. The required information includes workload specifications to describe the scenario and its usage intensity as well as software plans to describe the processing steps that execute.

### **3.2.1 Workload Specifications**

A workload specification consists of

- a description of a specific use of the system, and
- the workload intensity for each request

The most frequently used functions determine the overall performance of the system. Thus, early-lifecycle SPE workload descriptions focus on the most frequent uses of the system. The *workload description* includes the specific functions required for each use of the system being modeled and the order in which they are requested.

The *workload intensity* specifies the rate at which each use of the system being modeled is requested. For interactive systems, the intensity may be expressed either as the arrival rate for requests or, for multi-user systems, the number of concurrent users and the amount of time between their requests. For real-time systems, the intensity is described in terms of the arrival rate of the events that trigger and sustain the workload.

### **3.2.2 Software Plans**

The software plans describe the software execution path(s) for each workload. The software plans should specify the software components that execute, the order in which they execute, and any repetition as well as conditional and/or parallel execution of components for the corresponding workload.

Several different notations for describing software plans are available. Execution graphs [Smith, 1990 #480] have become a *de facto* standard among those who model performance. Execution graphs represent software components as nodes. A software component is a collection of program statements, procedures, and abstract machine calls that represents a logical function in the design. Transfer of control between components is represented by arcs.

## **3.3 Execution Environment**

The execution environment describes the platform on which the proposed system will execute. This virtual machine consists of a hardware configuration as well as the operating system and other software that interfaces with the proposed system.

### 3.3.1 Hardware Facility

The hardware facility is specified by identifying the key devices that are used to perform processing and their interconnections.

### 3.3.2 Service Times

Service times reflect the performance-related characteristics of the execution environment. For a processor, service time would normally be the amount of time required to execute an instruction. For I/O devices, one typically specifies the average time required to execute an I/O. Specifications for other types of devices depend on the device type and the type(s) of performance problems that can be expected. For example, with a network, one might be concerned with transmission speed, message processing overhead, or both.

### 3.4 Resource Requirements

Resource requirements estimate the amount of service required from key devices in the hardware configuration. Software plans typically specify resource requirements for processing steps in terms of the software resources (e.g., operating system calls or database accesses) that they use rather than primitive device services, such as CPU instructions. The processing overhead specification is then used to translate resource requirements into service time(s) on individual devices.

### 3.5 Processing Overhead

Processing overhead maps software resources onto device services. An overhead specification for a particular software resource type would list the devices used by that resource type and the amount of service required from each device. For example, a particular database access might require both some number of CPU instructions and several physical I/Os on a given disk.

## 4 SPE Meta-Model

From the above information, it is possible to derive a model of the information required to perform an SPE study. This model is known as the SPE meta-model since it is a model of the information that goes into constructing an SPE model.<sup>†</sup>

Note that this meta-model is different from the Performance Model Interchange Format (PMIF) discussed in [Smith, 1994 #1213]. The PMIF defines information exchanged between performance tools while the meta-model described here defines information to be exchanged between CASE and performance tools.

This section begins with a discussion of the techniques considered for formally representing the meta-model. The representation technique selected is then described. This is followed by a description of the SPE meta-model.

---

<sup>†</sup> The terminology surrounding multiple layers of models can be confusing to the uninitiated. A *model* contains some information. This information might, for example, be about the performance characteristics of some piece of software. A *meta-model* is a model of the information contained in a model; i.e., it models the model. A *meta-meta-model* is a model of the information contained in a meta-model.

## 4.1 Meta-Model Representation Techniques

Several different techniques for representing the SPE meta-model were considered. These include: entity-relationship models, class diagrams, and the EIA/CDIF approach. The relative merits of each of these is discussed briefly below.

### 4.1.1 Entity-Relationship Models

Entity-relationship (ER) models [Chen, 1976 #115] describe the static information structure of a problem by using entities to represent the principal abstractions of the problem. Binary relationships represent associations among those abstractions. Pure entity relationship models are somewhat restricted. They cannot represent hierarchical or taxonomic relationships between entities. In addition, relationships involving three or more entities must be decomposed and represented as binary relationships.

Entity-relationship models may be extended in a number of ways. Entity-relationship-attribute (ERA) models include attributes, which represent characteristics of entities. Other extensions include  $n$ -ary relationships, supertype/subtype relationships (generalization/specialization), and associative abstractions. Some modeling schemes also include an aggregation relationship which represents the situation where one entity is composed of instances of other entities.

A number of graphical representations have been used for ER and ERA models. Examples include: [Chen, 1977 #114], [Teorey, 1986 #520], [Blaha, 1988 #62], and [Shlaer, 1988 #475].

### 4.1.2 Class Diagrams

Class diagrams are used by most object-oriented methods to show the object classes that make up an application domain and the relationships among them. Class diagrams are essentially extended ERA diagrams that include supertype/subtype and aggregation relationships. Some approaches to constructing class diagrams also include associative objects and/or  $n$ -ary relationships. Others (e.g., [Booch, 1994 #1089]) include modeling constructs, such as meta-classes or parameterized classes, which are specific to object-oriented programming languages.

Again, several different graphical notations have been used for constructing class diagrams. The most widely known are the OMT (Object Modeling Technique) notation [Rumbaugh, 1991 #455] and the Booch notation [Booch, 1994 #1089].

### 4.1.3 EIA/CDIF Approach

This approach is described in the draft EIA/CDIF (Electronic Industries Association/CASE Data Interchange Format) standard [EIA, 1994 #1212]. CDIF is actually a family of standards that describe a mechanism for transferring information between CASE tools. The standards define a transfer format that allows tools that have different internal databases and storage formats to exchange information. An exchange takes place via a file and internal tool information is translated to and from the file's transfer format.

In the CDIF standard, the information to be transferred between two tools is known as a *model*. The contents of a model are defined using a *meta-model*. A meta-model defines

the information structure of a small area of CASE (such as data modeling or data-flow diagrams) known as a “Subject Area.” Each meta-model is, in turn, defined using a *meta-meta-model*. The meta-meta-model is based on the Entity-Relationship-Attribute (ERA) approach. It supports:

- binary relationships (ternary and higher-order relationships are not allowed)
- cardinality constraints on relationships (one-to-one, many-to-many, etc.)
- relationships with attributes
- associative entities
- subtyping or inheritance (including multiple inheritance)

The meta-meta-model also includes a graphical notation for describing meta-models. This notation supports the following concepts:

- entities: denoted by rectangles
- relationships: denoted by arrows
- relationship cardinality: indicated by a minimum:maximum annotation next to the appropriate entity symbol
- subtyping (inheritance): denoted by a line connecting the supertype(s) and subtype(s). (Note: A supertype must be drawn above its subtype(s).)

The CDIF meta-meta-model can be used to define a Software Performance Engineering meta-model. The SPE meta-model would define a CDIF “Subject Area.” A CDIF Transfer Format could then be used to define a standard format for transferring SPE information between tools.

## 4.2 Meta-Model Representation

The representation technique selected for this project combines the EIA/CDIF approach with object-oriented class diagrams. Basing the meta-model on the CDIF standard makes it possible to use one of the CDIF Transfer Formats to define an SPE interchange format in the future. Using an object-oriented notation overcomes several significant limitations of the CDIF graphical notation. The first is the lack of specific symbols for representing inheritance. Because CDIF does not have a symbol for inheritance, supertypes and subtypes must be indicated by placing the supertype above its subtype(s). This can make the diagram less readable and limits layout possibilities. A second limitation of the CDIF notation is the lack of a specific symbol for representing aggregation relationships. Although aggregation could be indicated using ordinary relationships with a label such as “IsComposedOf,” this relationship occurs frequently enough that it is advantageous to use a special symbol to denote it. Finally, the CDIF notation does not include a symbol for denoting associative entities. An associative entity represents a relationship between two other entities. While associative entities can often be identified by inspecting their attributes,<sup>§</sup> they occur frequently enough that it is advantageous to use a special symbol to denote them.

The graphical notation selected for this project is a subset of the OMT object-model notation developed by Rumbaugh, et. al. [Rumbaugh, 1991 #455]. This notation

---

<sup>§</sup> An associative entity will contain identifier attributes for each of the entities that it associates.



includes graphical syntax for inheritance as well as for composite and associative entities and, thus, overcomes the limitations encountered with the EIA/CDIF graphical notation. To maintain compatibility with the CDIF standard, however, we restrict relationships to be binary. The relevant elements of the OMT notation are summarized in Appendix A.

### 4.3 Meta-Model Description

This version of the meta-model defines the essential information required to create the software and system performance models as defined in <<Smith book>>. Additional information may be required for analysis of advanced system model features such as memory analysis, passive resource delays, petri net analysis, etc. The CDIF meta-model draft standard provides features for extending meta-model definitions to incorporate information supersets; the extensions are a topic for future work.

The meta-model ERA diagram is shown in Figure 1a. Figure 1b gives the attributes of each entity. The following paragraphs describe the entities and their relationships. The complete definition is in [Williams, 1994 #1245].

An SPE study is based on *PerformanceScenarios*. Each *PerformanceScenario* is modeled by an *ExecutionGraph*. An *ExecutionGraph* is composed of one or more *Nodes* and zero or more *Arcs*. A *Node* may be connected to one other *Node* via an *Arc*. Several types of *Nodes* may be used in constructing an *ExecutionGraph*:

- *ProcessingNode*: represents processing steps at some appropriate level of abstraction. There are three types of *ProcessingNodes*:
  - *BasicNode*: represents a software processing step at the lowest level of detail appropriate for the current development stage.
  - *ExpandedNode*: indicates that processing details are expanded in a subgraph at the next level of detail. The subgraph is, itself, another *ExecutionGraph*.
  - *LinkNode*: represents a functional component whose execution requirements are specified in a previously saved performance scenario.
- *StateIdentificationNode*: indicates lock-free and acquire-release processing events.
- *CompoundNode*: represents special processing structures, such as CASE constructs, repetition, and parallel execution. There are four types of *CompoundNode*:
  - *RepetitionNode*: represents components which are repeated, each with a repetition factor specifying the number of repetitions.
  - *CaseNode*: represents conditional execution of components, each with a probability of execution.
  - *PardoNode*: represents parallel execution paths, each with a probability of being initiated.

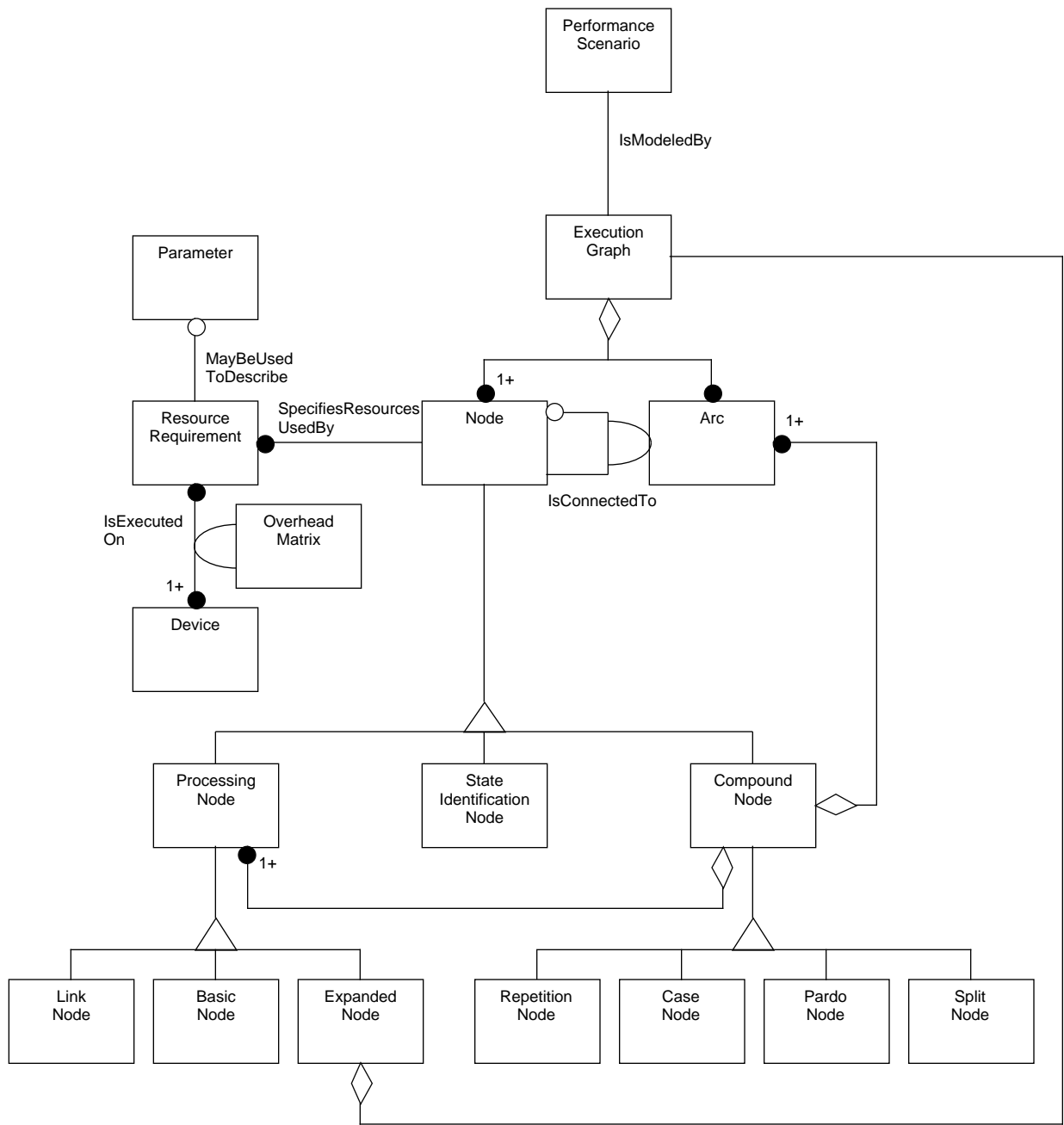


Figure 1a. SPE Meta-Model ERA Diagram

- *SplitNode*: indicates the initiation of concurrent processes that need not join.

A *CompoundNode* is also composed of a combination of one or more *ProcessingNodes* and one or more *Arcs*.

<b>Arc</b>	NodeList	Quantity	Description
ArcID	<b>CompoundNode</b>	SchedulingPolicy	ModificationDateTime
FromNode	CompoundNodeType	ServiceUnits	<b>ExpandedNode</b>
ToNode	<b>Device</b>	ServiceTime	NodeID
<b>BasicNode</b>	DeviceName	<b>ExecutionGraph</b>	<b>LinkNode</b>
<b>CaseNode</b>	DeviceType	GraphName	PerformanceScenarioID

<b>Node</b>	<b>Parameter</b>	InterarrivalTime	NodeID
NodeID	ParameterName	NumberOfJobs	ResourceName
NodeName	ParameterType	Priority	ServiceUnits
NodeType	ParameterValue	<b>ProcessingNode</b>	<b>SplitNode</b>
<b>OverheadMatrix</b>	<b>PardoNode</b>	ProcessingNodeType	<b>StateIdentificationNode</b>
ResourceName	NodeList	<b>RepetitionNode</b>	
DeviceName	<b>PerformanceScenario</b>	RepetitionFactor	
AmountOfService	ScenarioName	<b>ResourceRequirement</b>	

Figure 1b. Entity Attributes (Inherited attributes are not shown)

The resources used by a *Node* are specified by one or more *ResourceRequirements*. A *ResourceRequirement* may be described by an optional *Parameter*. A *ResourceRequirement* is executed on one or more *Devices*. A *Device* represents a unit that provides some processing service. *ResourceRequirements* are associated with *Devices* by an *OverheadMatrix* which specifies the amount of service that each resource type requires from various devices.

The current version of the meta-model does not include performance objectives. Currently, performance objectives are defined informally, based on the type of problem and expert judgement. Inclusion of performance objectives in the meta-model will require that they be more formally defined. This is a topic for future research.

The model is formally defined using the EIA/CDIF format. Figure 2 illustrates an entity definition, Figure 3 shows an attribute definition and Figure 4 illustrates a relationship definition.

---

META-ATTRIBUTE NAME.....**SchedulingPolicy**  
 CDIFMETAIDENTIFIER.....SPE046  
 DESCRIPTION.....The policy used to select the next service request to be served from a device queue.  
 USAGE.....  
 ALIASES.....  
 CONSTRAINTS.....  
 DATATYPE .....Enumerated  
 DOMAIN.....FCFS | PS | IS  
 LENGTH .....  
 ISOPTIONAL .....False

---

Figure 3. Sample Attribute Description

---

**ResourceRequirement.IsExecutedOn.Device**

NAME.....**IsExecutedOn**  
 CDIFMETAIDENTIFIER.....SPE060

---

**META-ENTITY: Device**

NAME.....**Device**  
 SUBTYPEOF.....  
 CDIFMETAIDENTIFIER.....SPE005  
 DESCRIPTION.....A Device represents a unit in the execution environment that provides some processing service.  
 USAGE.....  
 ALIASES.....  
 CONSTRAINTS.....  
 TYPE.....Kernel

---

INHERITED META-ATTRIBUTES ..

---

LOCAL META-ATTRIBUTES .....

DeviceName  
 DeviceType  
 Quantity  
 SchedulingPolicy  
 ServiceUnits  
 ServiceTime

---

INHERITED META-RELATIONSHIPS

---

LOCAL META-RELATIONSHIPS...

ResourceRequirement.IsExecutedOn.Device

---

Figure 2. Sample Entity Description

SUBTYPEOF.....	
SUPERTYPEOF.....	
MINSOURCECARD .....	0
MAXSOURCECARD .....	N
MINDESTCARD .....	0
MAXDESTCARD .....	N
DESCRIPTION.....	Relationship to connect a ResourceRequirement to its implementation on a particular Device or set of Devices. This relationship is described by the associative entity OverheadMatrix.
USAGE.....	
ALIASES.....	
CONSTRAINTS.....	

---

INHERITED META-ATTRIBUTES ..	
------------------------------	--

---

LOCAL META-ATTRIBUTES .....	
-----------------------------	--

---

Figure 4. Sample Relationship Definition




The entity *OverheadMatrix* merits some additional explanation. It is based on a concept in the SPE product, *SPE-ED™* [Smith, 1994 #1214]. The *OverheadMatrix* is an associative entity; it describes the relationship between a *ResourceRequirement* and a *Device*. An individual instance of *OverheadMatrix* contains an identifier for a *ResourceRequirement* (*ResourceName*) and a *Device* (*DeviceName*). For each *ResourceRequirement/Device* pair, the *OverheadMatrix* specifies a particular *AmountOfService*. For example, the *ResourceRequirement* may specify the number of instructions to be executed. The *OverheadMatrix* would specify the CPU processing time per instruction which results in the *AmountOfService* for the CPU *Device*. The identification of this entity as a matrix becomes clear if the entity type is viewed as a table with each instance corresponding to a row that specifies a distinct *ResourceRequirement/Device* pair such as:

- instructions and the CPU processing time per instruction,
- database updates and the CPU processing time per update and
- database updates and the Disk device visits per update.

The columns of the matrix contain the values of the attributes for each. Each node in the *PerformanceScenario* then specifies the number of Instructions and Database updates, and the overhead matrix relates the values to specific requirements for each *Device*. An example of an overhead matrix is in Figure 5, and Figure 6 shows the formal definition of the *OverheadMatrix* meta-entity.

**Software spec template:**  
 Overhead Demo

**Facility template:**  
 ATM Facility

Devices	CPU	ATM	DEV1
Quantity	1	1	1
Service units	Instr.	Screens	Phys I/

Ctl Flo	2		
CPU	1		
ATM	5	1	
I/O	5		1
Evt Rec	5		1

Service time	.00001	1	0.05
--------------	--------	---	------

Lock	Replace	Delete	Print
------	---------	--------	-------

Figure 5. Overhead Matrix Example

---

**META-ENTITY: *OverheadMatrix***

NAME.....OverheadMatrix  
SUBTYPEOF.....  
CDIFMETAIDENTIFIER.....SPE010  
DESCRIPTION.....The OverheadMatrix specifies the amount of service that  
each resource type requires from various devices.  
USAGE.....  
ALIASES.....  
CONSTRAINTS.....  
TYPE.....Associative

---

INHERITED META-ATTRIBUTES ..

---

LOCAL META-ATTRIBUTES .....

- ResourceName
- DeviceName
- AmountOfService

---

INHERITED META-RELATIONSHIPS

---

LOCAL META-RELATIONSHIPS...

---

**Figure 6. Definition of OverheadMatrix**

## **5 Summary and Conclusions**

This paper has presented a meta-model of the information requirements for Software Process Engineering. This meta-model defines the information required to perform early-lifecycle SPE. This definition can be used by CASE tool manufacturers to add the capability for capturing performance data to CASE tools that support software analysis and design methods. The meta-model also provides a basis for developing an interchange format to allow CASE tools to export performance information to performance modeling tools.

Outstanding issues for future work include:

- *Inclusion of performance objectives in the SPE meta-model:* As noted in Section 4, addition of performance objectives to the SPE meta-model requires their formalization.
- *Development of an SPE transfer format:* The SPE meta-model described here provides the basis for developing a transfer format for exchanging information between CASE tools and performance tools.
- *Reconciliation of models:* As noted in [Smith, 1994 #1213], SPE uses different levels of models. It is necessary to reconcile the high-level, early-lifecycle

model described in the SPE meta-model presented here with the more detailed model described in [Smith, 1994 #1213].

- *Inclusion of time in software methods*: Full integration of software methods and SPE requires that software methods be capable of directly representing timing and resource requirements. Approaches such as those proposed by Jahanian and Mok [Jahanian, 1987 #739] or Jaffe and Leveson [Jaffe, 1991 #628] show promise for bridging this gap.
- *Addition of performance results to the SPE meta-model*: The current SPE meta-model includes the information required to construct a software model. Adding performance results to the model would make it possible to transfer the results back to a CASE tool to support decisions on design alternatives.
- *Automated translation between CASE models and execution graphs*: One of the goals of this research is to provide support for evaluating the performance characteristics of software designs at an early stage in the development process. The SPE information model described defines the required information and a transfer format based on this model will make it possible to exchange information between CASE tools and performance tools.

The ability to easily compare design alternatives based on their performance characteristics also requires the ability to translate CASE models into performance models. We have identified scenarios [Booch, 1994 #1089], [Hsia, 1994 #1182], [Potts, 1994 #1183] as a common point of departure between software methods and SPE. Scenarios are central to the construction of SPE models. Scenarios are also becoming a central feature of contemporary requirements analysis and software design methods. The use of interaction diagrams [Booch, 1994 #1089] appears to be particularly promising [Williams, 1994 #1245].

## 6 References

[Baldassari, et al., 1989]

M. Baldassari, B. Bruno, V. Russi, and R. Zompi, "PROTOB: A Hierarchical Object-Oriented CASE Tool for Distributed Systems," *Proceedings of the European Software Engineering Conference, 1989*, Coventry, England, 1989.

[Blaha, et al., 1988]

M. R. Blaha, W. J. Premerlani, and J. E. Rumbaugh, "Relational Database Design Using an Object-Oriented Methodology," *Communications of the ACM*, vol. 31, no. 4, pp. 414-427, 1988.

[Booch, 1994]

G. Booch, *Object-Oriented Analysis and Design with Applications*, Redwood City, CA, Benjamin/Cummings, 1994.

[Buhr, et al., 1989]

R. J. A. Buhr, G. M. Karam, C. J. Hayes, and C. M. Woodside, "Software CAD: A Revolutionary Approach," *IEEE Transactions on Software Engineering*, vol. 15, no. 3, pp. 235-249, 1989.

[Chen, 1977]

P. Chen, "The Entity Relationship Approach to Logical Data Base Design," Monograph No. 6, Q. E. D. Information Sciences, Inc., 1977.



- [Chen, 1976]  
P. P. Chen, "The Entity Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9-36, 1976.
- [EIA, 1994]  
EIA, "CDIF – CASE Data Interchange Format Overview," EIA/IS-106, Electronics Industries Association, January, 1994.
- [Hsia, et al., 1994]  
P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, and C. Chen, "Formal Approach to Scenario Analysis," *IEEE Software*, vol. 11, no. 2, pp. 33-41, 1994.
- [Jaffe, et al., 1991]  
M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart, "Software Requirements Analysis for Real-Time Process Control Systems," *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 241-258, 1991.
- [Jahanian and Mok, 1987]  
F. Jahanian and A. K.-L. Mok, "A Graph-Theoretic Approach for Timing Analysis and its Implementation," *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 961-975, 1987.
- [Lor and Berry, 1991]  
K. Lor and D. M. Berry, "Automatic Synthesis of SARA Design Models from System Requirements," *IEEE Transactions on Software Engineering*, vol. 17, no. 12, pp. 1229-1240, 1991.
- [Martin, 1988]  
C. R. Martin, "An Integrated Software Performance Engineering Environment," Masters Thesis, Duke University, 1988.
- [Opdahl, 1992]  
A. Opdahl, "A CASE Tool for Performance Engineering During Software Design," *Proceedings of the Fifth Nordic Workshop on Programming Environmental Research*, Tampere, Finland, 1992.
- [Opdahl and Sølvsberg, 1992]  
A. Opdahl and A. Sølvsberg, "Conceptual Integration of Information System and Performance Modeling," *Proceedings of the Working Conference on Information System Concepts: Improving the Understanding*, 1992.
- [Potts, et al., 1994]  
C. Potts, K. Takahashi, and A. I. Anton, "Inquiry-Based Requirements Analysis," *IEEE Software*, vol. 11, no. 2, pp. 21-32, 1994.
- [Rolia, 1992]  
J. A. Rolia, "Predicting the Performance of Software Systems," Ph.D. Thesis, University of Toronto, 1992.
- [Rumbaugh, et al., 1991]  
J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Englewood Cliffs, NJ, Prentice Hall, 1991.
- [Shen, et al., 1990]  
V. Shen, C. Richter, M. Graf, and J. Brumfield, "VERDI: A Visual Environment for Designing Distributed Systems," Technical Report No. STP-054-90, Microelectronics and Computer Technology Corporation, January, 1990.
- [Shlaer and Mellor, 1988]  
S. Shlaer and S. J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data*, Englewood Cliffs, NJ, Yourdon Press, 1988.

- [Smith, 1994]  
C. U. Smith, "Definition Of A Performance Model Interchange Format," *Performance Engineering Services*, October, 1994.
- [Smith, 1990]  
C. U. Smith, *Performance Engineering of Software Systems*, Reading, MA, Addison-Wesley, 1990.
- [Smith and Williams, 1990]  
C. U. Smith and L. G. Williams, "Why CASE Should Extend into Software Performance," *Software Magazine*, vol. 10, no. 9, pp. 49-65, 1990.
- [Smith and Wong, 1994]  
C. U. Smith and B. Wong, "SPE Evaluation of a Client/Server Application," *Proceedings of the Computer Measurement Group*, Orlando, FL, 1994.
- [Teorey, et al., 1986]  
T. J. Teorey, D. Yang, and J. P. Fry, "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model," *ACM Computing Surveys*, vol. 18, no. 2, pp. 197-222, 1986.
- [Valderruten, et al., 1992]  
A. Valderruten, O. Hijiej, A. Benzekri, and D. Gazal, "Deriving Queueing Networks Performance Models from Annotated LOTOS Specifications," *Proc. 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, R. Pooley and J. Hillston, ed., Edinburgh, 1992, pp. 167-178.
- [Williams, 1994]  
L. G. Williams, "Definition of Information Requirements for Software Performance Engineering," Technical Report No. SERM-021-94, Software Engineering Research, October, 1994.
- [Woodside, 1989]  
C. M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks," *Performance Evaluation*, vol. 9, 1989.
- [Woodside, et al., 1991]  
C. M. Woodside, E. M. Hagos, E. Neron, and R. J. A. Buhr, "The CAEDE Performance Analysis Tool," *Ada Letters*, vol. XI, no. 3, 1991.

## Appendix A: OMT Object Model Notation

The OMT Object Model Notation [Rumbaugh, 1991 #455] is used to document the classes in an application and the relationships among them. This report uses a subset of this notation to graphically document the SPE meta-model. The subset used here was chosen for conformance with the EIA/CDIF proposed standard for CASE data interchange [EIA, 1994 #1212]. The graphical symbols used to construct the SPE meta-model are shown in Figure A.1.

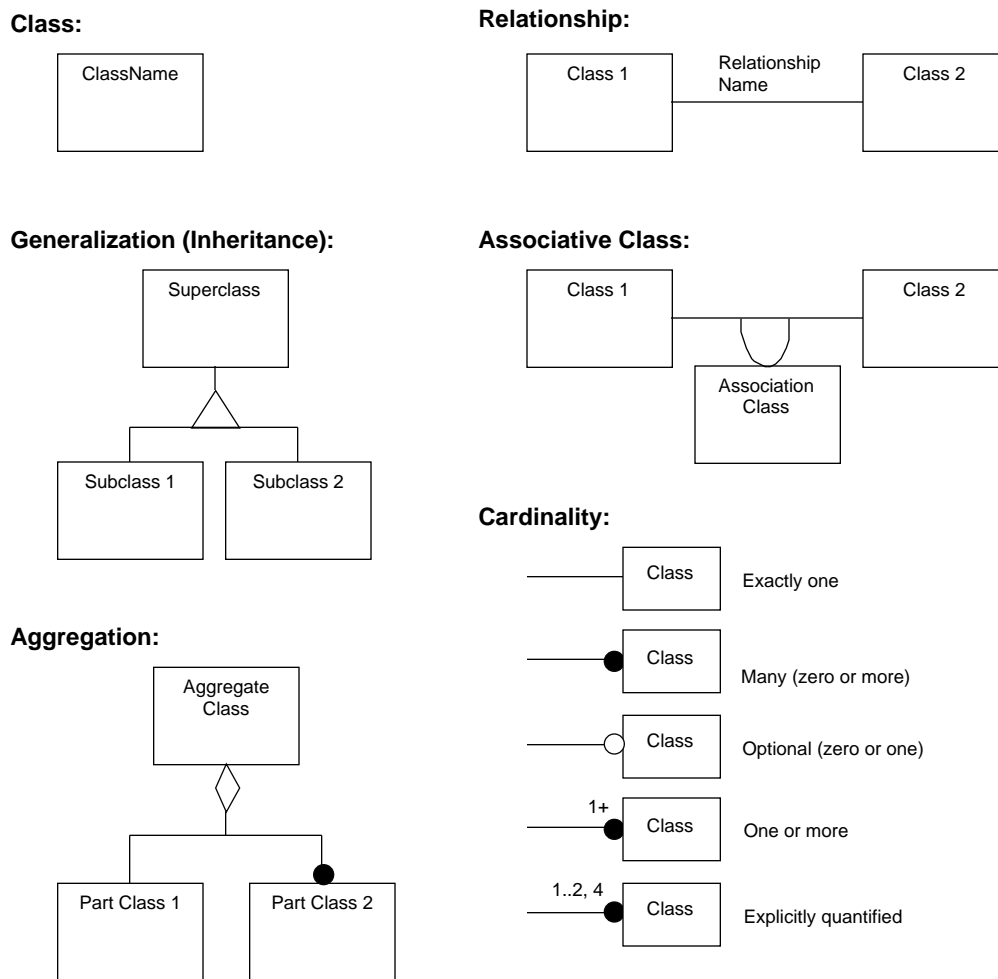


Figure A.1: OMT Object Model Notation Symbols

A class is denoted by a rectangle labeled with the class name. The attributes of the class may optionally be included in the rectangle. A class models a meta-entity in the CDIF format. For classes with more than a few attributes, however, this makes the diagram difficult to read. Here, attributes are included only in the textual description of the meta-model.

A relationship between two classes is indicated by a line connecting the two classes.<sup>†</sup> The line is labeled with the name of the relationship. Cardinality constraints on

<sup>†</sup> The OMT notation allows ternary relationships. However, to conform with the EIA/CDIF draft standard, we have restricted relationships to be binary only.

relationships are indicated by decorations on the line next to the class whose participation they constrain. The various decorations are shown in Figure A.1.

Inheritance is modeled by a generalization relationship. With inheritance, properties common to a group of classes are assigned to a *superclass*. Each *subclass* inherits all of the attributes and relationships of its superclass(es). A generalization relationship is indicated by a triangle whose apex points at the superclass.

Associative classes model relationships as classes. This allows adding information and behavior to the relationship. Each instance of the relationship becomes an instance of the associative class. Associative classes correspond to associative meta-entities in the CDIF format. While the proposed CDIF standard included associative meta-entities, the CDIF graphical notation does not include special syntax to indicate these entities. An associative class is indicated by a loop connecting the associative class to the relationship that it represents.

Aggregation models a whole/part or “is composed of” relationship in which an instance of one class is composed of instances of one or more other component, or part, classes. The proposed CIDF standard does not include a special representation for aggregation. However, aggregation is a commonly used relationship and it is useful to be able to indicate it directly on the graphical model. In the OMT notation, aggregation is indicated by a diamond attached to the aggregate class.