A Framework for Performance Engineering during Information System Development

Andreas L. Opdahl* and Arne Sølvberg†
The University of Trondheim
andreaso@ifi.unit.no, asolvber@idt.unit.no

Abstract

Software performance engineering aims at predicting and improving the performance of applications during development and in production. This paper presents a framework for performance engineering of information systems with emphasis on parameter estimation support.

The need for performance engineering of information systems is discussed. Views of information system and performance modelling are presented. It is shown how application specifcations can be extended with performance annotations. The resulting framework is applied to predict and improve the performance of projected applications during development. Sensitivity analysis is supported to point out performance bottlenecks in the application and suggest which parameters to estimate with most care. Target platform modelling is provided to relieve the information system developer from assessing the performance of the target platform and operating system software. The framework is realised in terms of the PrM language for software specification. Finally, some conclusions are offered.

Introduction

Software performance engineering [Smi90, AWS91] aims at predicting and improving the performance of software during development and in production. Efficient utilisation of information technology has become a decisive competition factor in industry and business. Although the price of computers is steadily decreasing, the total hardware expenses are *increasing* along with the demand for computing power.

*Department of Informatics, Faculty of Nature Sciences, College of Arts and Sciences
†Information Systems Group, Department of Electrical Engineering and Computer Science, The Norwegian Institute of Technology

As a result, hardware expenses have become visible at the organisation level, and therefore a limited resource. At the same time, the price of unacceptable performance has grown due to tightened market competition. New methods must be applied to utilise the organisation's computer resources more efficiently.

This paper will present a framework for performance engineering of information systems. The framework focuses on predicting the performance of projected applications during development. It has been developed with the goals of 1) integrating the state of the art of information system development and performance engineering [OS92]; 2) supporting performance parameter estimation [Opd91b]; 3) interacting with the capacity management process [Vet91], and 4) supporting database design [OS81]. In addition, simplicity and generality has been emphasised. This paper will focus on parameter estimation support in particular.

Although the framework is presented in a theoretical form, it has been developed as a result of practical considerations and experience. A realisation of the framework has been made in connection with the experimental integrated CASE tool environment PPP [GLW91]. A graphical interface for annotating PPP specifications with performance parameters has been implemented, together with the basic associated analysis techniques [Opd92b]. The tool has been developed as part of the IMSE[1] project, which provides state-of-the-art performance modelling tools at the computer system level [HP89]. A case study has been undertaken, using the framework and tool to monitor the development of a commercial information system [BOVS91].

Views of information system and performance modelling are presented in sec. 1, and application specifcations are annotated with performance parameters. The resulting framework is applied to predict and improve the performance of projected applications during development in sec. 2. Sensitivity analysis is supported to point out performance bottlenecks in the application and suggest which parameters to estimate with most care in sec. 3. Target platform modelling is provided to relieve the information system developer from assessing the performance of the target platform and operating system software in sec. 4 The framework is realised in terms of the PrM language for software specification in sec. 5. Finally, some conclusions are offered.

In the remainder of this paper, important terms are **bold-faced** when introduced.

1 The basic framework

The basic framework for performance engineering during information system development is based on general views of information system and performance modelling, as well as on annotating application specifications with performance parameters.

[1]The IMSE project was a collaborative research project supported by the CEC as ESPRIT project no 2143. It was carried out by the following organisations :- BNR Europe STL, Thomson CSF, Simulog A.S., University of Edinburgh, INRIA, IPK (Berlin), University of Dortmund, University of Pavia, SINTEF (University of Trondheim), University of Turin and University of Milan.

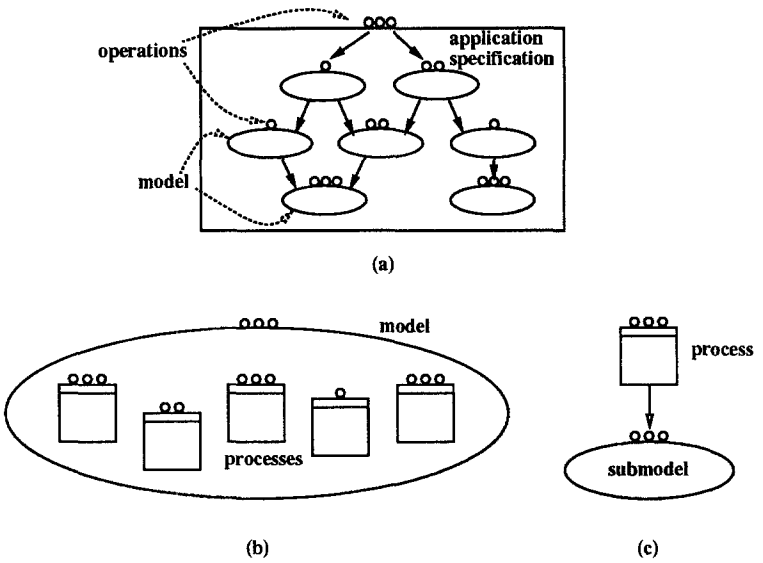

Figure 1: An application specification contains a DAG of models (a). A model contains a set of processes (b). A non-primitive process has a submodel (c).

The three are treated in separate.

1.1 Information system modelling

Fundamental to the basic framework is the view of *information system modelling* presented in [OS92]. According to this view, an **application specification** S represents the **projected application** during development. No assumption is made about the manner in which this specification is established, only that it exists in a consistent state every time a **performance prediction** is to be made. An application specification provides **operations** o resembling how the projected application will provide **functions** to the organisation using it. Since the projected application is likely to be complex, the specification is hierarchical, comprising a DAG (directed acyclic graph) of *models* (fig. 1a).[2]

A **model** m is a composite, dynamic, partial view of an application. A model provides a set of operations, just like an application specification. It is composite because it contains a set of *processes* (fig. 1b).[3] Models may have **submodels**, and models which are not submodels of any other models are **top-level models**. The set

[2]The framework has been developed in the context of dataflow diagram modelling, in which *cycles* in submodel graphs are uncommon. Allowing cycles is a possible future extension of this work.

[3]Of course, most models will contain constructs other than "processes," e.g. "stores" and "flows" in the dataflow based class of languages. However, these constructs are not relevant

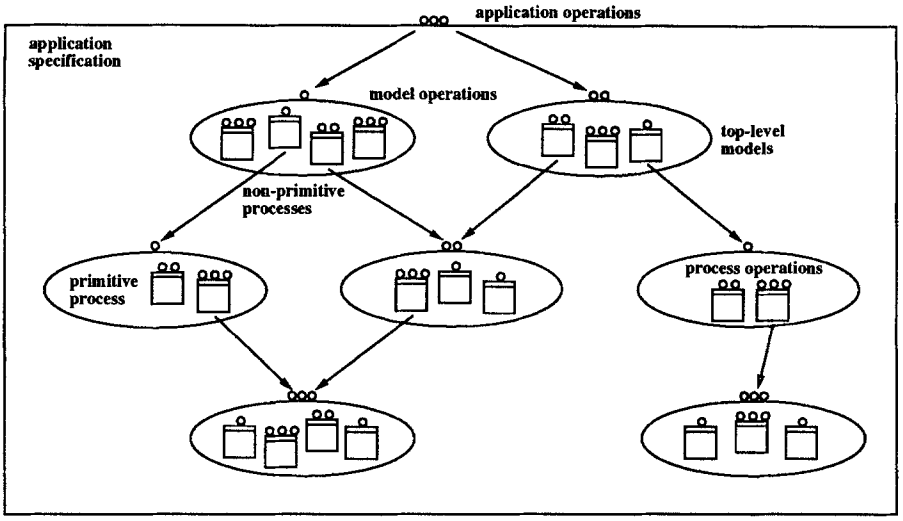

Figure 2: Overall picture of the information system view.

of operations provided by an application specification corresponds to the operations provided by its top-level models.

A **process** p is an atomic, dynamic, partial view of an application. A process provides a set of operations, just like an application specification or a model. A process is atomic because it is not a composition of something else in the way models are.[4] Note that the only difference between the definitions of model and process is that the former is composite while the latter is atomic. This means that a process at one level in the decomposition graph may correspond to a model at the next (fig. 1c). Such a model is the **submodel** of the process. Processes with no submodels associated with them (yet) are **primitive**. Other processes are **non-primitive**.

The operations provided by a model are the **external operations** of that model. The operations provided by the processes contained in a model are its **internal operations**.

Fig. 2 shows an overall picture of the information system view of this section.

1.2 Performance modelling

Complementary to the information system view of sec. 1.1 is the view of *performance modelling* also presented in [OS92]. According to this view, a **performance model** M abstracts a computer system. A performance model provides **service centers** s to

within the framework, which considers only the *dynamics* of the application specification.

[4] However, processes may have submodels containing other processes *associated with* them.

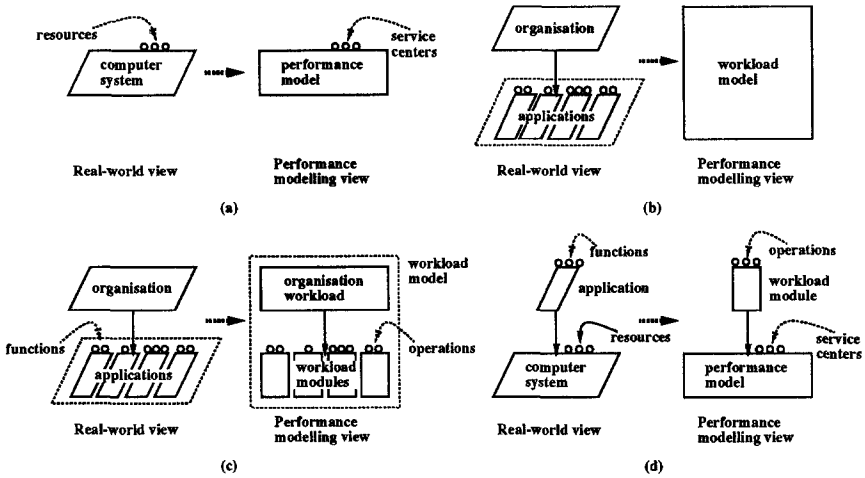

Figure 3: Performance modelling views of performance model (a), workload model (b), organisation workload (c), and workload module (c-d).

resemble how the computer system provides **resources** to the applications running on it (fig. 3a). Examples of such resources are CPU's, disks, communication channels, and other hardware devices. The aim of performance modelling is to obtain a **performance prediction** through analysis of a performance model under a *workload model*, just like the computer system exhibits a certain **performance** under some **workload**.

A **workload model** W abstracts the projected and existing applications that (will) run on a computer system and the organisation which uses (will use) them (fig. 3b). A workload model consists of an *organisation workload* and a set of *workload modules* (fig. 3c).

An **organisation workload** O abstracts how often an organisation uses the functions provided by the available applications. An organisation workload is a set of **operation intensities**, which abstract how often an organisation uses a specific function provided by an application. An operation intensity is either *transaction*, *interactive*, or *batch* [LZGS84]. The set of operation intensities contained in the organisation workload corresponds to the set of functions provided by the applications and used by the organisation.

A **workload module** w abstracts how the functions provided by a projected or existing application use the resources provided by the computer system (fig. 3d). A workload module provides operations to the organisation workload and uses the service centers of the performance model. The set of operations provided by a workload module corresponds to the set of functions provided by the application it abstracts, just as for application specifications. The workload module of an application specification S is represented as a **module demand matrix** $\mathbf{D}^{ws}$ of average service center

uses. Each row of this matrix represents an operation provided by the workload module, and each row element represents a number of uses of a service centre of the performance model.

1.3 Annotating application specifications

To interface information system with performance modelling, the information system view of sec. 1.1 must be extended so that a *workload module* $\mathbf{D}^{ws}$ can be derived from an application specification S. For this purpose, three kinds of performance annotations are needed [OS92]. These are the *initiation descriptions* and the *execution descriptions* for models, and the *demand descriptions* for primitive processes. Each such description corresponds to annotations that must be added to the application specification prior to performance analysis.

Initiation and execution descriptions together specify how many uses of each internal operation of some model that correspond to one use of each of its external operations. First, the initiation description specifies on average which internal operations are used initially when each external operation is used. Then, the execution description specifies on average which internal operations are used next when each internal operation itself is used.

An **initiation description** abstracts how each external operation on some model initiates uses of its internal operations (fig. 4a). This defines the initial state of the model for each of the operations it provides. The initiation description of a model m is represented as an **initiation matrix** $\mathbf{I}^m$ of average initial internal operation uses. Each row of this matrix represents an external operation of the model, and each row element represents an initial number of uses of an internal operation. The framework requires initiation descriptions for all models in the application specification.

An **execution description** abstracts how each internal operation of some model next lead to uses of other of its internal operations (fig. 4b). This defines the state transitions of the model. The execution description of a model m is represented as an **execution matrix**[5] $\mathbf{A}^m$ of average next internal operation uses. Each row of this matrix represents an internal operation of the model, and each row element represents a number of subsequent uses of an internal operation of the model. The framework requires execution descriptions for all models in the application specification. This means that creating both initiation and execution descriptions must be simple for a realisation of the framework to be successful.

In addition to relating internal and external operations of models to one another, a relation is needed between the operations of primitive processes and the service centers of the performance model. A **demand description** abstracts how each operation of a process or model requires uses of the service centers of the performance model (fig. 4c). At this stage, we are only interested in the demand descriptions of primitive processes.

[5]Note that this execution matrix is *not* a Markov matrix (or transition probability matrix) [TK84], as its elements abstract transitions between *processes* rather than between *states*.

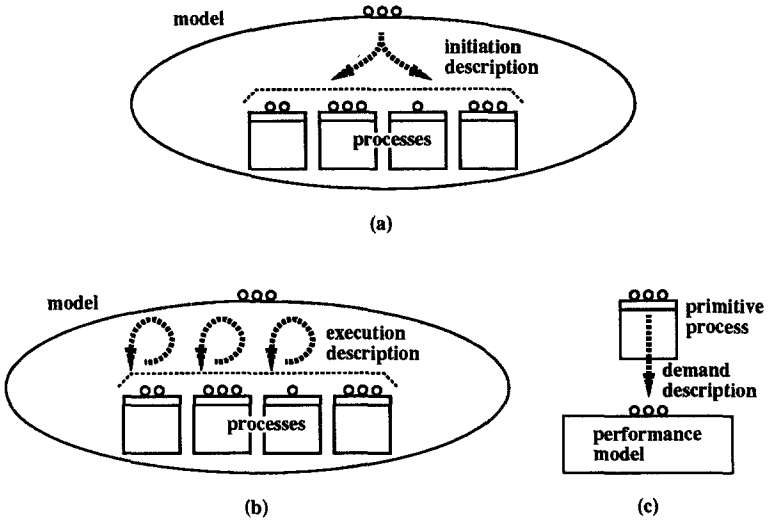

Figure 4: The initiation description relates uses of external operations to the internal operations used initially (a), the execution description relates uses of internal operations to the internal operations used next (b), and the demand description relates uses of operations of primitive processes to uses of performance model service centers (c).

The demand description of a primitive process p is represented as a **primitive demand matrix** $\mathbf{D}_p^m$ of average service center uses, where m is the model containing p. Each row of this matrix represents an operation provided by the process, and each row element represents a number of uses of a service centre of the performance model. The framework requires demand descriptions for all primitive processes in the application specification. This means that demand descriptions will be created for close to every process of the application specification, since every process is primitive at some point of application development. Therefore, creating demand descriptions must also be easy for a realisation of the framework to be successful.

2 Performance prediction

The goal of the basic framework is to predict the performance of projected applications during development. This section demonstrates how a workload module can be derived from an annotated application specification, before considering *performance analysis* and the *performance measures* it produces.

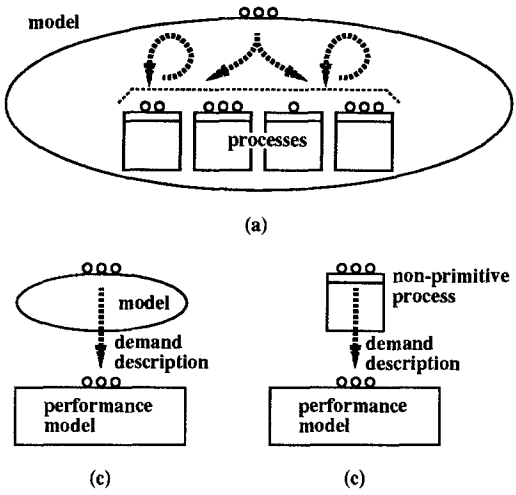

Figure 5: The operation count relates uses of external operations to uses of internal operations (a), while the demand descriptions also relates uses of operations of models (b) and non-primitive processes (c) to uses of performance model service centers.

2.1 Workload module derivation

The three kinds of performance annotations presented in sec. 1.3 facilitate automatically deriving a workload module from an annotated application specification S. *Workload module derivation* proceeds through bottom-up collapsing of its decomposition graph, starting with the lowest level of models. When all the submodels of model m have been collapsed, an **operation demand matrix** $\mathbf{D}^m_{\mathcal{O}_m}$ is derived by stacking all its **process demand matrices** on top of one another in the appropriate order,

$$\mathbf{D}^m_{\mathcal{O}_m} = \begin{array}{c} \overline{\mathbf{D}^m_{p_1}} \\ \vdots \\ \underline{\mathbf{D}^m_{p_n}} \end{array}, \tag{1}$$

since each internal operation of model m (represented by a row of $\mathbf{D}^m_{\mathcal{O}_m}$) corresponds to an operation provided by some process p_i (represented by a row of $\mathbf{D}^m_{p_i}$). An **operation count matrix** $\mathbf{C}^m$ for model m specifies how many times each internal operation is used when each of its external operations are being used. It is derived as (fig. 5a) [Low73, OS81]

$$\mathbf{C}^m = \mathbf{I}^m(1 - \mathbf{A}^m)^{-1}, \tag{2}$$

where $\mathbf{1}$ is the identity matrix, since the total number of internal operations used (represented by $\mathbf{C}^m$) equals the sum of operations used initially (represented by $\mathbf{I}^m$)

and subsequently (represented by $\mathbf{C}^m\mathbf{A}^m$) so that

$$\mathbf{C}^m = \mathbf{I}^m + \mathbf{C}^m\mathbf{A}^m.$$

For the sensitivity analysis of sec. 3 we also define a **process count matrix** $\mathbf{C}_p^m$ for process p of model m, which specifies how many times each internal operation of model m provided by process p is used when each of its external operations are used. It is directly derived as the corresponding subset of columns of the operation count matrix, since each column of $\mathbf{C}^m$ corresponds to an internal operation provided by one of its processes.

A **model demand matrix** $\mathbf{D}^m$ for the model is then calculated as (fig. 5b)

$$\mathbf{D}^m = \mathbf{C}^m\mathbf{D}^m_{\mathcal{O}_m}, \tag{3}$$

since the total number of service centre uses $\mathbf{D}^m$ is determined by 1) the number of service centre uses of its internal operations (represented by the rows of $\mathbf{D}^m_{\mathcal{O}_m}$) and 2) how many times each internal operation is used (represented by the columns of $\mathbf{C}^m$). If m is the submodel of process p of model m', its **non-primitive process demand matrix** $\mathbf{D}_p^{m'}$ is derived as (fig. 5c)

$$\mathbf{D}_p^{m'} = \mathbf{D}^m,$$

since the operations provided process p corresponds to the external operations of its submodel m (both represented by the rows of $\mathbf{D}_p^{m'}$ and $\mathbf{D}^m$). This prepares for collapsing of model m' at the next-higher level of decomposition. When all the top-level models of application specification S have been collapsed in this way, its **module demand matrix** $\mathbf{D}^{ws}$ can be derived by stacking all its top-level **model demand matrices** on top of one another in the appropriate order,

$$\mathbf{D}^{ws} = \begin{array}{c} \overline{\mathbf{D}^{m_1}}, \\ \vdots \\ \underline{\mathbf{D}^{m_n}}. \end{array}$$

again since each operation provided by the specification (represented by the rows of $\mathbf{D}^{ws}$) corresponds to an operation provided by a top-level model of the specification (represented by the rows of $\mathbf{D}^{m_i}$).

Unless the computer system will be dedicated to the projected application, additional workload modules must be created for each existing application running on the computer system. Workload modules representing existing applications are similar to those representing projected ones. All these additional workload modules, as well as the corresponding organisation specifications, must be established by conventional means [LZGS84, Fer78].

2.2 Performance analysis

A **performance analysis** of the computerised information system can be undertaken as soon as all the workload modules necessary have been derived or established. The

performance model used in this analysis must be created by conventional means. In some cases, the performance model will already have been created for capacity management purposes, or as a result of previous performance engineering efforts. In a wider sense, performance models may be queueing networks, Petri-net models, or written in special- or general-purpose simulation languages. Within the basic framework, however, mean-value analysis (MVA) [LZGS84] of separable queueing network models has been assumed for simplicity. Each operation provided by the workload modules is represented as a *customer class* in this analysis. The corresponding row of the module demand matrix, together with its operation intensity, becomes a customer description of the class.

2.3 Performance measures

[LZGS84] identifies two groups of **performance measures** output from this analysis: 1) **responsiveness measures**, for the provided and existing applications which are either a) response times $r^{o,S}$ for terminal operations, b) throughputs $x^{o,S}$ for batch operations, or c) residence times $r^{o,S}$ for transaction operations o of application specification S, and 2) **service centre measures**, for the computer system which are either a) utilisations $u^{s,M}$ or b) sojourn times $z^{s,M}$ for service centre s of performance model M.

In this manner, performance predictions are obtained for the computerised information system *before* the projected application is in production. The responsiveness measures for the projected application indicate whether it will perform clearly acceptable or clearly unacceptable, or if further analyses based on refined application specifications are needed. These measures may also be used to compare the performance of alternative designs. The responsiveness measures for the existing applications predict to what extent their performance will be degraded by the projected one. If the performance of some projected or existing application is unacceptable, precautions can be taken before the projected application in put in production. Service centre utilisations for the performance model indicate where bottlenecks will be located. This information is useful if the computer system must be upgraded, or in case some application must be optimised. In the latter case, its module demand matrix predicts which resources it will use most heavily.

A service centre **sojourn time** predicts how much time *a single use* of the corresponding computer system resource will take on average. The **sojourn time vector** Z^M for performance model M is defined as

$$Z^M = \langle z^{1,M}, \ldots, z^{S_M,M} \rangle. \tag{4}$$

These sojourn times will become important in sec. 3.

Performance analyses should be carried out several times throughout application development. The workload module representing the projected application must be derived anew every time the application specification has been changed. However, the operation intensities, workload modules for existing applications, as well as the performance model, can be reused. This considerably reduces the effort involved.

3 Sensitivity analysis

The basic framework of sec. 1 requires numerous parameters. Secs. 3 and 4 will extend the framework to support *estimation* of performance parameters.

Sensitivity analysis on the residence times of a model is useful for determining: 1) which parameters that are most crucial to performance, thus focusing design and code optimising effort, and 2) which parameters that must be estimated with most care and which parameters are less important, thus focusing parameter capture effort accordingly. Sensitivity analysis is either: 1) *local*, deriving the sensitivity of a model residence time on parameters of that model, or 2) *global*, deriving the sensitivity of a model residence time on parameters of its direct or indirect submodels. We will treat the two in separate, after introducing the concept of residence times of models. This section applies methods developed in [OS81], and extends them to cover hierarchical modelling.

3.1 Residence times

A **residence time vector** R^m for model m specifies how much time is spent when each of its external operations are being used. It is derived as

$$R^m = \mathbf{D}^m Z^M, \tag{5}$$

since each row of $\mathbf{D}^m$ represents the number of times each performance model service centre is used by model m when each of its external operations are used, and each element of Z^M represents *how much time* is spent per service centre use. By insertion of eqs. 3 and 2 into eq. 5, we arrive at

$$R^m = \mathbf{I}^m(\mathbf{1} - \mathbf{A}^m)^{-1}\mathbf{D}^m_{\mathcal{O}_m} Z^M, \text{ and} \tag{6}$$

$$r^{o,m} = (I^{o,m})^T(\mathbf{1} - \mathbf{A}^m)^{-1}\mathbf{D}^m_{\mathcal{O}_m} Z^M, \tag{7}$$

for the residence time $r^{o,m}$ of operation o on model m, where $I^{o,m}$ is the o'th row vector of $\mathbf{I}^m$ and $r^{o,m}$ is the o'th element of R^m. These equations define the residence times of a model in terms of its initiation and execution descriptions, and in terms of the demand descriptions of its processes.

3.2 Local sensitivity analysis

The sensitivities of the residence time of operation o on model m can now be determined by differentiation of eq. 7. The sensitivities of residence time $r^{o,m}$ on initiation row vector $I^{o,m}$, on execution matrix $\mathbf{A}^m$, and on operation demand matrix $\mathbf{D}^m_{\mathcal{O}_m}$ become

$$\frac{\partial r^{o,m}}{\partial I^{o,m}} = (\mathbf{1} - \mathbf{A}^m)^{-1}\mathbf{D}^m_{\mathcal{O}_m} Z^M, \tag{8}$$

$$\frac{\partial r^{o,m}}{\partial \mathbf{A}^m} = [(\mathbf{1} - \mathbf{A}^m)^{-1}]^T I^{o,m}(Z^M)^T(\mathbf{D}^m_{\mathcal{O}_m})^T[(\mathbf{1} - \mathbf{A}^m)^{-1}]^T, \text{ and} \tag{9}$$

$$\frac{\partial r^{o,m}}{\partial \mathbf{D}^m_{\mathcal{O}_m}} = [(\mathbf{1} - \mathbf{A}^m)^{-1}]^T I^{o,m} (Z^M)^T, \tag{10}$$

where the non-trivial derivation of $\frac{\partial r^{o,m}}{\partial \mathbf{A}^m}$ is outlined in appendix A. Appendix B accordingly defines (trivial) *vector forms* of eqs. 8–10 which are necessary in sec. 3.3.

These vector equations provide all local parameter sensitivities on $r^{o,m}$ with a minimum of calculation. A local sensitivity analysis for model m implies calculating the sensitivities of all its parameters and sorting the results. Top parameters in the sorted list are most important to the model residence time, while middle and lower rank parameters can be regarded as unimportant. In this way, sensitivity analysis not only suggests which parameters to estimate with most care, but also points out *where* design and code optimisations will have the most impact, as already mentioned.

3.3 Global sensitivity analysis

The equations of sec. 3.2 defined the sensitivities of the residence times r^{o,m_1} for model m_1 on all its parameters. However, these residence times may also depend on parameters of the direct and indirect submodels m_n of model m_1. This case calls for *global* sensitivity analysis, as opposed to local.

Let r^{o,m_1} be the residence time for operation o on model m_1 and let x^{m_n} be a scalar parameter of one of its direct or indirect submodels m_n. Assuming that the decomposition graph of the application specification is a *tree*[6], the sensitivity of residence time vector $R^{m_1} = \langle r^{o_1,m_1}, \ldots, r^{o_n,m_1} \rangle$ on x^{m_n} is derived as [Opd91b]

$$\frac{\partial R^{m_1}}{\partial x^{m_n}} = C^{m_1}_{p_1} C^{m_2}_{p_2} \cdots C^{m_{n-1}}_{p_{n-1}} \frac{\partial R^{m_n}}{\partial x^{m_n}}, \tag{11}$$

where $\frac{\partial R^{m_n}}{\partial x^{m_n}}$ is one of the vector forms[7] of eqs. 8, 9, and 10 defined in appendix B, and the process count matrices $\mathbf{C}^{m_i}_{p_i}$ were defined in sec. 2.1, and model m_{i+1} decomposes process p_i of model m_i, $i = 1 \ldots n-1$.

An exhaustive global sensitivity analysis for model m implies calculating the sensitivities of all its parameters, as well as all the parameters of its direct and indirect submodels, and sorting the results. Again, top parameters in this list are most important to the model residence times, while middle and lower rank parameters can be regarded as unimportant.

An important special case of global sensitivity analysis is the **overall sensitivity analysis**, which is an exhaustive global sensitivity analysis for all the top-level models of an application specification. Although the equations of this section have been designed to be computationally efficient, the cost of performing an exhaustive sensitivity analysis may become prohibitive due to the possibly large number of parameters involved. Methods to reduce the search space should be sought for.

[6] [Opd91b] also considers the more general case where the decomposition graph is a DAG.

[7] ...depending on whether x^{m_n} is an initiation, execution, or demand matrix element.

The simplest approach is of course to restrict the scope of a global sensitivity analysis to 1) certain kinds of parameters or 2) certain levels of decomposition only. [Opd91b] also extends the framework with *residence time analysis* to provide some of the sensitivity analysis support at lower computational cost.

3.4 Performance model sensitivity analysis

A single parameter of the annotated application specification is not likely to have major impact on the overall workload generated by the application. Therefore, the performance model sojourn time vector Z^M of eq. 4 can be regarded as constant during sensitivity analysis. However, if this assumption does not hold, the Z^M vector will vary slightly when application specification parameters are changed. In particular, this may be the case for initiation and execution matrices of higher-level models in the specification. A forthcoming paper [Opd92a] will address this topic in detail, providing *exact* sensitivity analysis for combined software and hardware performance models. Hence, it must be kept in mind that the sensitivities calculated from eqs. 8–11 are only *approximations* in such cases.

4 Target platform modelling

Sec. 3 introduced sensitivity analysis to support performance parameter estimation, based on the assumption that some parameters *already* had been estimated. This section makes parameter estimation easier in the first place by extending the basic framework with *target platforms*.

So far, we have annotated application specifications with resource demand estimates in terms of computer system resources. This creates two main problems:

- Today there is a tendency to build software by putting together existing **components** [Søl90, Vet91]. Common examples of such components are high-level languages (HLL's), database management systems (DBMS's), and screen handlers constituting the **target platform software** of the projected application. This means that an application is specified in terms of target platform functions rather than in terms of resources at the computer system level.

  This makes the framework difficult to apply because: 1) application developers are not used to think in terms of computer system resources, and it is difficult for them to annotate application specifications in terms of such low-level concepts, and 2) assessing the performance of the target platform software, e.g. an optimised relational database management system, is a very specialised task.

- Every application running on a computer system induces operating system overhead on that computer system. Common examples of such overhead are the workloads of virtual memory managers, dispatchers, and interrupt handlers constituting the **operating system software** of the computer system.

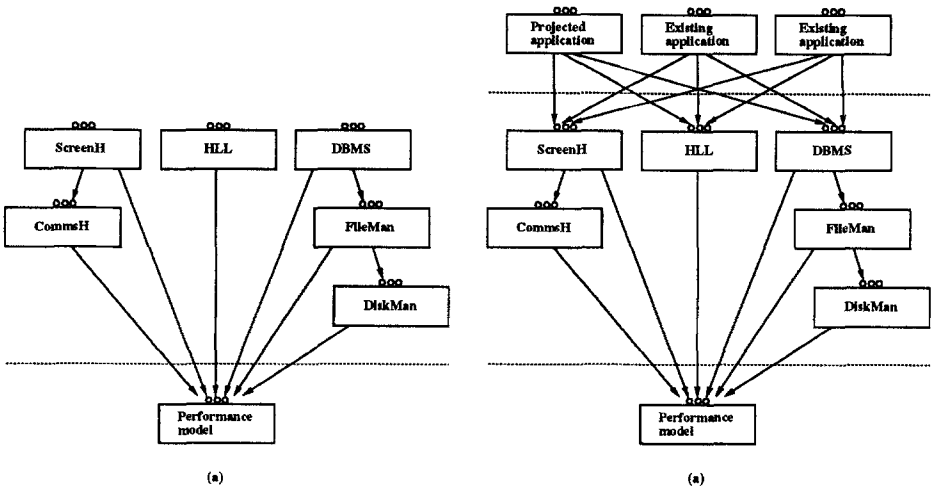

Figure 6: A target platform model (a), and composite workload model (b).

This means that the resource annotations of an application specification must include estimates of the induced operating system overhead for the derived workload module to be correct.

Again, this makes the framework difficult to apply because: 1) application developers are not used to take the performance of the operating system software into consideration when designing applications, and 2) the operating system overhead of an application is not fixed for that application, but a function of the *total* workload on the computer system. This workload is produced both by the projected application *and* by other, already existing applications.

All the above difficulties conflict the aim of making performance engineering closely integrated with information system development.

4.1 The target platform model

The framework therefore introduces *target platform modelling* to avoid the above problems. A **target platform model** P abstracts the target platform and operating system software of a computer installation. A target platform model provides operations, just like application specifications, models, and processes. Since the target platform and operating system software is likely to be complex, the target platform model is again hierarchical, comprising a DAG of workload modules (fig. 6a).[8]

As in the basic framework, we derive a workload module from the annotated application specification, and establish additional workload modules for each existing

[8]This view of workload modelling is based on Hughes' *sp* approach [Hug88].

application by conventional means. The resulting workload model therefore consists both of 1) top-level application workload modules and 2) lower-level target platform workload modules (fig. 6b). In contrast to the workload models of sec. 1.2, a **composite workload model** W thus comprises a DAG (as opposed to a *set*) of workload modules. The target platform model (without workload modules representing applications), is itself a composite workload model in this sense.

In this way, target platform modelling resolves the problems stated at the beginning of this section:

- The higher-level target platform workload modules correspond to the target platform software components that the projected application will be implemented upon, typically a high-level language (HLL), a database management system (DBMS), and a screen handler. Each top-level target platform module provides operations that the application workload modules use. Thus instead of annotating the primitive processes of the application specification in terms of performance model service centre uses, we supply them with demand descriptions in terms of target platform operations.
- The lower-level target platform workload modules correspond to the operating system software components of the computer system, such as the virtual memory manager, the dispatcher, and the interrupt handler. Thus instead of having to include estimates of the induced system software workload in the demand descriptions, the overhead is automatically calculated during the *workload model analysis* of sec. 4.2.

4.2 Workload model analysis

The composite workload model resulting from target platform modelling must be analysed prior to performance analysis. Analysis proceeds through bottom-up collapsing of its module graph with repeated matrix multiplications, much as in the workload derivation of sec. 2.1. The outcome is a collapsed **module demand matrix** $\mathbf{D}^w$ for each application workload module w of the workload model. These matrices constitute the workload modules representing the projected and existing applications. The workload modules are applicable for performance analysis according to sec. 2.2.

4.3 Interface with sensitivity analysis

Apart from the additional workload model analysis prior to performance analysis, target platform modelling does not require modifications of the basic framework. However, to combine the sensitivity analysis technique of sec. 3 with target platform modelling, we need to redefine the concept of residence times of operations on models. Sec. 3.1 defined these times in terms of service centre sojourn times. Since target platform modelling replaces the concept of performance model service centres with target platform operations, we need to introduce the new concept of

platform estimates $z^{o,P}$ representing the sojourn time of operation o provided by target platform P. Again, a target platform operation sojourn time predicts how much time *a single use* of the corresponding target platform function will take on average.

To provide such platform estimates we first need perform a *target platform analysis*. Target platform analysis proceeds exactly like the workload model analysis of sec. 4.2, with the top-level application workload modules removed. The outcome is a **platform demand matrix** $\mathbf{D}^P$ for target platform P.

We now define the **platform estimate vector** $Z^P = \langle z^{1,P}, \ldots, z^{O_P,P} \rangle$ for target platform P as

$$Z^P = \mathbf{D}^P Z^M, \tag{12}$$

since 1) each row of $\mathbf{D}^P$ represents the number of times each performance model service centre is used when each target platform operation is used, and 2) each element of Z^M represents *how much time* is spent per service centre use.

Sensitivity analysis now proceeds with platform estimates $z^{o,P}$ and Z^P replacing the computer system resource sojourn times $z^{s,M}$ and Z^M. This substitution, however convenient for the purpose of sensitivity analysis, is not obvious. [Opd91b] presents two alternative techniques for validating this *platform estimate assumption*.

4.4 Database optimisation

The critical point in establishing a target platform model, is representing the optimised database management system (DBMS) as a workload module. [BO91, BOVS91] discusses alternative parameter capture strategies for an SQL-type DBMS in the context of a practical case study. Wieland [Wie91] has established and validated a workload module specification for simple queries on the INGRES relational database system. This is a topic for further research.

5 A realisation of the framework

This paper has presented a framework for performance engineering during information system development based only on a minimal set of requirements about the development methodology and modelling language used. Isolating the framework in a linear algebra representation, as has been done, ensures its *generality*. In principle, it can be interfaced with any operationally oriented and hierarchic methodology for information system development, and with any performance evaluation method. To demonstrate the *practicalness* of the approach, a realisation of the framework has been made in connection with the IMSE environment for performance evaluation and the experimental integrated CASE tool PPP.

The IMSE is an *i*ntegrated *m*odelling *s*upport *e*nvironment for performance evaluation. It focuses on 1) *availability* of state-of-the-art performance evaluation meth-

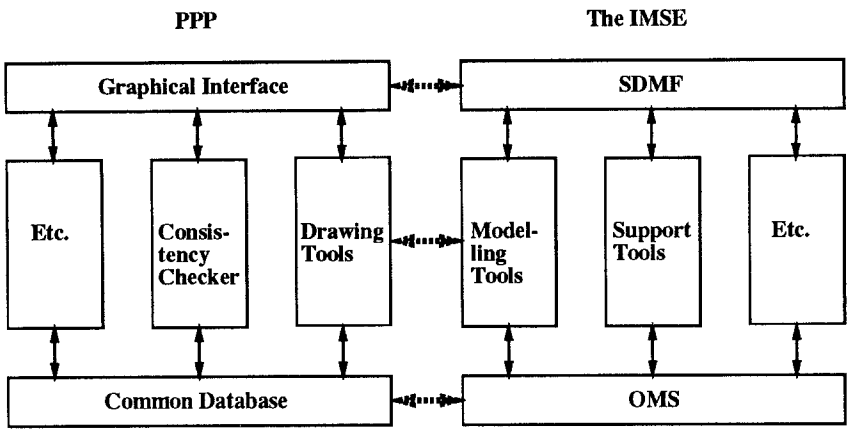

Figure 7: The performance modelling and analysis support provided by the IMSE made available to the PrM tool user.

ods through easy-to-use graphical interfaces, and 2) *integrating* and *supporting* the performance evaluation methods through a common set of utilities. The IMSE builds on existing performance evaluation tools for workload derivation [CS85], workload modelling [Hug86], queueing networks [Pot85], Petri-net models [BC89], and special-purpose simulation [PB88]. In addition, the IMSE provides a set of environmental tools supporting execution of static and dynamic performance models, animation of model executions, automated support for planning and performing experiments, and generation of reports from experiments. The IMSE tools share a *support environment* containing a graphical user interface system and a common object management system.

PPP (*p*rocesses, *p*henomena, and *p*rograms) is an experimental integrated CASE tool for information system development [GLW91]. It focuses on 1) *formality* to facilitate early verification and validation as well as automated model-to-model translation and code-generation, and 2) *integration* between the modelling tools used, and between different phases of development.[9] PPP builds on well-known approaches to information system development, such as top-down design and the DFD and ER paradigms.

In particular, the *PrM language* [BCSA86] of PPP is a formalised extension of dataflow diagrams, avoiding several of their impreciseneses by introducing 1) *flow of control* as well as data flow; 2) *triggering* and *termination* to define process dynamics; 3) *port connectives* to define what is consumed and produced by each process per execution, and 4) *operations* to distinguish between different ways of triggering a process or a model. A *PrM tool* has been implemented as a realisation of the framework of this paper. Fig. 7 depicts how the IMSE and PPP environments

[9] And between the problem analysis and design phases in particular.

can be integrated through the PrM tool, which lets PrM specifications be annotated with performance parameters. The tool automatically generates IMSE workload models from annotated application specifications, using the algorithm of sec. 2.1.

Conclusions

The need for performance engineering of information systems was discussed. Views of information system and performance modelling were presented in sec. 1, and application specifcations were extended with performance annotations. The resulting framework was applied to predict and improve the performance of projected applications during development in sec. 2. Sensitivity analysis was supported to point out performance bottlenecks in the application and suggest which parameters to estimate with most care in sec. 3. Target platform modelling was provided to relieve the information system developer from assessing the performance of the target platform and operating system software in sec. 4 The framework was realised in terms of the PrM language for software specification in sec. 5.

In the introduction, the goals of the framework were stated as 1) integrating the state of the art of information system development and performance engineering; 2) supporting performance parameter estimation; 3) interacting with the capacity management process, and 4) supporting database design.

Concerning the first one, [OS92] has presented a conceptual integration of information system and performance modelling. This paper has focused on meeting the second of the goals. Furthermore, [Opd91b] extends the framework with additional techniques for a) *residence time analysis* which suggests which parts of the application to annotate; b) *bounds analysis* as an alternative to average analysis early in the design, and c) *parametric analysis* in case obtaining one or a few parameters is infeasible. The third goal of relating capacity management to the framework is discussed in [Vet91, BOVS91], while database design has been treated in [OS81]. Thus while work remains on the latter two, two of the goals have been met so far. Furthermore, a tool has been implemented to support the framework [Opd92b], and has been applied in a practical case study [BO91].

Composite workload models for the organisation's software systems can be used to balance workload between the available computers while controlling application response times. Alternative hardware configurations can be evaluated and compared. Workload modules for projected applications can be combined with models of the existing ones, and the hardware resources can be extended at the right moment. The performance of the projected application is continuously monitored throughout development. Good resource utilisation is ensured and it is made clear where the design and code may be improved.

The framework is an advancement on contemporary approaches to performance engineering of information systems in the areas focused on. The framework gains on its simplicity, generality, and emphasis on parameter estimation support. The most important advancement, however, is integrating the information system and perfor-

mance modelling fields at the conceptual level and at the tool level. This facilitates bringing together recent advances in performance modelling with integrated CASE tools for information system development.

The complexity of contemporary information systems will continue to increase along with the organisation's dependency on them. Controlled, tool-supported management of the computer resources will therefore give the organisation an increasing competitive edge in the future.

Acknowledgement

We would like to thank the edp-personnel at The Regional Hospital in Trondheim, Tandem Computers in Trondheim and Oslo, and Twinco in Oslo for their continuing support and supply of information throughout this work.

References

[AWS91] Reda A. Ammar, J. Wang, and H. A. Sholl. Graphic modelling technique for software execution time estimation. *Information and Software Technology, Vol. 33, No. 2, Butterworth-Heinemann Ltd.*, March, 1991.

[BC89] Gianfranco Balbo and Giovanni Chiola. Stochastic Petri net simulation. Technical report, University of Turin, 1989.

[BCSA86] S. Berdal, S. Carlsen, A. Sølvberg, and R. Andersen. Information system behaviour expressed through process port analysis. Technical report, Division of Computer Science, The Norwegian Institute of Technology, 1986.

[Ber84] Margaret E. Berry. The best of both worlds: An integrated approach to capacity planning and software performance engineering. *Proc. Computer Measurement Group Conference XV, San Fransisco*, pages 462–466, Dec. 1984.

[BO91] Gunnar Brataas and Andreas L. Opdahl. Deriving workload models of projected software: A case study. Technical report, IMSE Project Report R6.6 – 9, Version 1, The Norwegian Institute of Technology, October 23, 1991.

[BOVS91] Gunnar Brataas, Andreas L. Opdahl, Vidar Vetland, and Arne Sølvberg. Information systems: Final evaluation of the IMSE. Technical report, IMSE Project deliverable D6.6 – 2, Version 1, SINTEF (Unversity of Trondheim), December, 1991.

[CS85] M. Calzarossa and G. Serazzi. A software tool for the workload analysis. *Proceedings from 'Modelling Techniques for Performance Analysis'*, 1985.

[Fer78] Domenico Ferrari. *Computer Systems Performance Evaluation.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1978.

[GLW91] Jon Atle Gulla, Odd Ivar Lindland, and Geir Willumsen. PPP — An integrated CASE environment. *Proceedings of "CAiSE91, Trondheim, Norway"*, May 1991.

[HP89] Peter H. Hughes and Dominique Potier. The integrated modelling support environment (ref. r1.2-3 ver. 1). *Presented at the ESPRIT-Week, Brüssels*, 1989.

[Hug83] Peter H. Hughes. A structural analysis of information processing systems (with applications to the sizing problem). Technical report, no. 28/83, Division of Computer Science, The Norwegian Institute of Technology, June, 1983.

[Hug86] Peter H. Hughes. Notes on system structure and performance specification. Technical report, The Norwegian Institute of Technology, April, 1986.

[Hug88] Peter Hughes. *sp* principles. Technical report, STC Technology Ltd. o59/ICL226/0, July 1988.

[LOS88] Odd Ivar Lindland, Andreas L. Opdahl, and Guttorm Sindre. PPM — The process & phenomenon model. *Proc' Infotech '88, Oslo*, 1988.

[Low73] T. C. Lowe. Analysis of an information system model with transfer penalties. *IEEE Trans. Comput. C-22, pp. 469-480*, 1973.

[LZGS84] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 07632, 1984.

[Opd91a] Andreas L. Opdahl. Deriving workload models of projected software: Basic framework. Technical report, IMSE Project Report R6.6 – 8, Version 2, The Norwegian Institute of Technology, October 23, 1991.

[Opd91b] Andreas L. Opdahl. Deriving workload models of projected software: Parameter estimation support. Technical report, IMSE Project Report R6.6 – 10, Version 1, The Norwegian Institute of Technology, October 23, 1991.

[Opd92a] Andreas L. Opdahl. Sensitivity analysis of combined software and hardware performance models. Not yet published, 1992.

[Opd92b] Andreas L. Opdahl. A CASE tool for performance engineering during software design. *Proceedings of "The Fifth Nordic Workshop on Programming Environment Research", Tampere/Finland*, 8–10 January, 1992.

[OS81] Harald Oftedahl and Arne Sølvberg. Data base design constrained by traffic load estimates. *Information Systems, Vol. 6, No. 4, pp. 267–282,* 1981.

[OS92] Andreas L. Opdahl and Arne Sølvberg. Conceptual integration of information system and performance modelling. *Proceedings of IFIP WG 8.1 Working Conference on: "Information Systems Concepts: Improving the Understanding", Alexandria/Egypt,* 13–15 April, 1992.

[PB88] R. J. Pooley and M. W. Brown. A diagramming paradigm for hierarchical process oriented discrete event simulation (ref. csr-254-88). Technical report, University of Edinburgh, January 1988.

[Pot85] Dominique Potier. New users introduction to QNAP2. Technical report, Rapport technique 40, INRIA, 1985.

[San77] John Winston Sanguinetti. Performance prediction in an operating system design methodology. Technical report, Ph.D. Thesis, University of Michigan, 1977.

[SB75] Howard A. Sholl and Taylor L. Booth. Software performance modelling using computation structures. *IEEE Transactions on Software Engineering, Vol. SE-1, No. 4, December,* 1975.

[Smi80] Connie U. Smith. The prediction and evaluation of the performance of software from extended design specifications. Technical report, Ph. D. dissertation, Department of Computer Sciences, The University of Texas at Austin, August, 1980.

[Smi90] Connie U. Smith. *Performance Engineering of Software Systems.* Addison-Wesley Publishing Company, 1990.

[Søl90] Arne Sølvberg. Integrated modelling and support environments for information systems. *Paper presented at the 23rd Newcastle-upon-Tyne International Seminar on the Teaching of Computing Science at University Level,* 1990.

[TK84] Howard M. Taylor and Samuel Karlin. *An Introduction to Stochastic Modeling.* Academic Press, Inc., 1984.

[Vet91] Vidar Vetland. Deriving composite workload models of existing software. Technical report, IMSE Project Report R6.6 – 7, Version 1, The Norwegian Institute of Technology, 1991.

[Wie91] Peter Wieland. Performance modelling and performance measurements of a relational dtabase management system. Technical report, Diploma Thesis, The Norwegian Institute of Technology, The University of Trondheim, 1991.

A Deriving execution matrix derivates

The sensitivity of $r^{o,m}$ on execution matrix $\mathbf{A}^m$ becomes

$$\frac{\partial r^{o,m}(\mathbf{A}^m)}{\partial \mathbf{A}^m} = \frac{r^{o,m}(\mathbf{A}^m + \mathbf{dA}^m) - r^{o,m}(\mathbf{A}^m)}{\mathbf{dA}^m}.$$

We have [OS81]

$$\begin{aligned} r^{o,m}(\mathbf{A}^m + \mathbf{dA}^m) &= (I^{o,m})^T(\mathbf{1} - \mathbf{A}^m - \mathbf{dA}^m)^{-1}\mathbf{D}^m_{\mathcal{O}_m} Z^M \\ &= (I^{o,m})^T[(\mathbf{1} - \mathbf{A}^m)(\mathbf{1} - (\mathbf{1} - \mathbf{A}^m)^{-1}\mathbf{dA}^m)]^{-1}\mathbf{D}^m_{\mathcal{O}_m} Z^M \\ &= (I^{o,m})^T(\mathbf{1} - (\mathbf{1} - \mathbf{A}^m)^{-1}\mathbf{dA}^m)^{-1}(\mathbf{1} - \mathbf{A}^m)^{-1}\mathbf{D}^m_{\mathcal{O}_m} Z^M. \end{aligned}$$

We set

$$\mathbf{Q}^m = (\mathbf{1} - \mathbf{A}^m)^{-1}$$

and get

$$\begin{aligned} r^{o,m}(\mathbf{A}^m + \mathbf{dA}^m) &= (I^{o,m})^T(\mathbf{1} - \mathbf{Q}^m\mathbf{dA}^m)^{-1}\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M \\ &= (I^{o,m})^T[\sum_{i=0}^{\infty}(\mathbf{Q}^m\mathbf{dA}^m)^i]\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M \\ &= (I^{o,m})^T[\mathbf{1} + \mathbf{Q}^m\mathbf{dA}^m + (\mathbf{Q}^m\mathbf{dA}^m)^2 + \cdots]\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M \\ &= (I^{o,m})^T\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M + (I^{o,m})^T\mathbf{Q}^m\mathbf{dA}^m\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M \\ &= (I^{o,m})^T(\mathbf{1} - \mathbf{A}^m)^{-1}\mathbf{D}^m_{\mathcal{O}_m} Z^M + (I^{o,m})^T\mathbf{Q}^m\mathbf{dA}^m\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M \\ &= r^{o,m}(\mathbf{A}^m) + (I^{o,m})^T\mathbf{Q}^m\mathbf{dA}^m\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M. \end{aligned}$$

This means that

$$\begin{aligned} \frac{\partial r^{o,m}(\mathbf{A}^m)}{\partial \mathbf{A}^m} &= \frac{r^{o,m}(\mathbf{A}^m + \mathbf{dA}^m) - r^{o,m}(\mathbf{A}^m)}{\mathbf{dA}^m} \\ &= \frac{(I^{o,m})^T\mathbf{Q}^m\mathbf{dA}^m\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M}{\mathbf{dA}^m} \\ &= [(I^{o,m})^T\mathbf{Q}^m]^T[\mathbf{Q}^m\mathbf{D}^m_{\mathcal{O}_m} Z^M]^T \\ &= (\mathbf{Q}^m)^T I^{o,m}(Z^M)^T(\mathbf{D}^m_{\mathcal{O}_m})^T(\mathbf{Q}^m)^T \\ &= [(\mathbf{1} - \mathbf{A}^m)^{-1}]^T I^{o,m}(Z^M)^T(\mathbf{D}^m_{\mathcal{O}_m})^T[(\mathbf{1} - \mathbf{A}^m)^{-1}]^T. \end{aligned}$$

Q.E.D.

B The vector forms of eqs. 8–10

The vector forms of eqs. 8–10 are defined as

$$\begin{aligned}
\frac{\partial R^m}{\partial i^{o',m'}_{o'',p''}} &= \langle \frac{\partial r^{1,m}}{\partial i^{o',m'}_{o'',p''}}, \ldots, \frac{\partial r^{o,m}}{\partial i^{o',m'}_{o'',p''}}, \ldots, \frac{\partial r^{O_m,m}}{\partial i^{o',m'}_{o'',p''}} \rangle, \\
\frac{\partial R^m}{\partial a^{m'}_{(o',p'),(o'',p'')}} &= \langle \frac{\partial r^{1,m}}{\partial a^{m'}_{(o',p'),(o'',p'')}}, \ldots, \frac{\partial r^{o,m}}{\partial a^{m'}_{(o',p'),(o'',p'')}}, \ldots, \frac{\partial r^{O_m,m}}{\partial a^{m'}_{(o',p'),(o'',p'')}} \rangle, \text{ and} \\
\frac{\partial R^m}{\partial d^{o',p'}_{s,M}} &= \langle \frac{\partial r^{1,m}}{\partial d^{o',p'}_{s,M}}, \ldots, \frac{\partial r^{o,m}}{\partial d^{o',p'}_{s,M}}, \ldots, \frac{\partial r^{O_m,m}}{\partial d^{o',p'}_{s,M}} \rangle,
\end{aligned}$$

where

$$\frac{\partial r^{o,m}}{\partial i^{o',m'}_{o'',p''}}$$

is element (o'', p'') of vector $\frac{\partial r^{o,m}}{\partial I^{o,m}}$ defined in eq. 8;

$$\frac{\partial r^{o,m}}{\partial a^{m'}_{(o',p'),(o'',p'')}}$$

is element $[(o', p'), (o'', p'')]$ of the matrix $\frac{\partial r^{o,m}}{\partial \mathbf{A}^m}$ defined in eq. 9, and

$$\frac{\partial r^{o,m}}{\partial d^{o',p'}_{s,M}}$$

is element $[(o', p'), (s, M)]$ of the matrix $\frac{\partial r^{o,m}}{\partial \mathbf{D}^m_{O_m}}$ defined in eq. 10.