Using the Tools in TRADE, I: A Decision Support System for Traffic Light Maintenance

S.F.M. van Vlijmen* R.J. Wieringa†

November 10, 1997

Contents

*Faculty of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, vlijmen@phil.ruu.nl.
†Faculty of Mathematics and Computer Science, Free University, De Boelelaan 1081a, 1081 HV Amsterdam, roelw@cs.vu.nl, http://www.cs.vu.nl/∼roelw

Abstract

In this paper we develop a specification of requirements and conceptual design of a decision support system for the maintenance of the lamps in traffic regulation systems. Requirements and design are both specified using semiformal techniques in TRADE (Toolkit for Requirements and Design Engineering). The decision support system is non-trivial and provides an existing need in a company that works in the traffic system management area. The TRADE specification is based upon an existing algebraic specification in PSF (Process Specification Formalism). The rationale for doing this is that the PSF specification is very detailed and lacks a proper requirements specification. It turns out that TRADE, at least in this case, is much better suited as a means to specify abstract requirements. We discuss how algebraic specification could fit in a TRADE framework.

1 Introduction

An important part of the maintenance of traffic light is lamp replacement. According to the Dutch firm *Nederland Haarlem*, it is not yet well-understood how lamp replacement can be best organized. This observation was the motivation for starting a joint project of the University of Utrecht, CWI[1] and *Nederland Haarlem*, with as goals the specification of lamp replacement policies that can be optimized on economic efficiency and reliability, and the specification of a decision support system that implements the policies (van der Duyn Schouten et al. 1996). The current report is a successor study, in which informal and semiformal specifications for the decision support system are developed that complement the formal specification given by van der Duyn Schouten et al. (1996).

We define a traffic light installation, or TLI for short, as the whole of light installations that serves to control traffic streams at a road junction. "Junction" should here be taken liberally. Lights that are geographically close, i.e., bridging the distance between two of them is not a significant effort in time and money, can be joined in the same TLI, and nevertheless belong to different traffic regulation systems.

For maintenance activities like lamp replacement, two quantities are of main importance: the economic costs of the maintenance, and the reliability of the TLI. Good policies are 'cheap' at the one hand, and yield a 'high' reliability at the other hand. Though it may seem an contrived problem, the issues that play a role are not that specific. A TLI can be classified as a multi component system with identical, or at least comparable, components. These are found in many occasions, e.g., lighting of offices and streets, machine parks in manufacturing, and series of military equipment. In the literature many policies for these systems are studied and have been developed, see Cho & Parlar (1991) and Dekker et al. (1997) for overviews. In this context, a policy is a replacement procedure. Such a policy is in general accompanied by a collection of index numbers that are fixed by following a optimization procedure. The joint project with *Nederland Haarlem* resulted in the following.

- First, five policies were identified. Two of these policies were described and studied in detail, i.e., a mathematical model was developed that gives expressions for exploitation costs and reliability per unit of time in terms of the index numbers of the policies. These policies are called the Indirect Class Grouping (ICG) and the Opportunity Based Replacement (OBR). These will be discussed in Section 2.

- Second, for ICG and OBR optimization procedures were developed. The procedures were implemented in Pascal.

- Third, using the optimization procedures, a small and rudimentary prototype was developed.

[1] CWI is the National Research Institute for Mathematics and Computer Science.

- Finally, a specification, in PSF (Mauw & Veltink 1989, Mauw & Veltink 1990, Mauw & Veltink 1993, Veltink 1993, Veltink 1995), was written that describes the major datatypes of a decision support system and its main functions.

The PSF specification is sufficiently detailed to be used as a means to elicit further functional demands by running it as a prototype. Nevertheless, the specification lacks some important descriptions and definitions. The requirements part of the specification is weak. In fact, it jumps from a vague concept right into a detailed algebraic specification.

Algebraic specification turned here out not to be really suited for, at least, the notation of the initial steps leading gradually from a goal to a system decomposition at the level of an implementation. Algebraic specification forces one to write quite detailed descriptions, this is dangerous when the concepts described are still shaky, as in the case at hand. Furthermore, algebraic specification tends to blurr intuitions that might be there, because of its lack of notational efficiency and elegancy.

The motivation of this paper is, first, to find out whether the TRADE technique is better suited than algebraic specification for a requirements specification of the system. Second, we discuss whether an algebraic specification may fit in a TRADE framework, as the means for further system decomposition, as a later stage, following a TRADE specification.

TRADE is based upon an analysis of structured and object-oriented software specification methods (Wieringa 1995, Wieringa & Saake 1996, Wieringa 1996, Wieringa 1997*a*, Wieringa 1997*b*). An earlier version was known as MCM (Wieringa 1993). The current version has a preliminary definition (Wieringa 1997*c*) but will be defined more elaborately after a number of case studies, including this one, are finished.

The rest of the paper is structured as follows. In Section 2, a short introduction is given to the two basic policies for lamp replacement as described in (van der Duyn Schouten et al. 1996). The idea is here to explain the concepts that appear in later specifications, not to discuss the operational research intricacies. Sections 3 and 4 give a TRADE specification of external interactions and a conceptual decomposition of a decision support system for traffic light replacement. Section 5 concludes the report with a discussion of possible extensions and of the role of formal specification in requirements specification.

2 Replacement Policies

The simplest replacement policy is **Laissez faire**, which consists solely of the immediate replacement of a lamp when it is defect. All other replacement policies add some more intelligence to the replacement process in the form of a replacement schemes for preventive maintenance. Two of these are ICG and OBR. Before we discuss the ICG and OBR policies, we review the parameters used by both these policies to determine a replacement scheme.

2.1 Parameters

Let a TLI contain $n \geq 1$ light points. Parameters of light points are subscripted with i, $1 \leq i \leq n$. The ICG and OBR policies assume that the set of light points has been partitioned into **groups**. For a reliable match between groups and their representation, it is preferred that groups contain light points that have comparable "replacement characteristics", i.e., failure behavior and replacement costs. One parameter of the ICG and OBR policies is, then, the partitioning of all light points into groups. We assume that each TLI is partitioned in an integer number of groups $m \geq 1$. Parameters of groups are subscripted with j, $1 \leq j \leq m$. Each group belongs to exactly one TLI but one TLI can contain more than one group.

We distinguish two kinds of replacement: If all lamps in one group are replaced, even if some of the lamps are not defect, then we call this replacement **preventive**. If an individual lamp is replaced

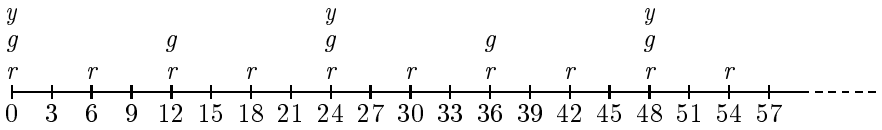

Figure 1: The preventive replacement of the groups r, g, and y in time.

because it is defect, we call this replacement **corrective**. So preventive replacements are always group replacements, and corrective replacements are always individual replacements. Note that preventive replacements may involve replacement of individual lamps that are defect!

The following parameters are used to determine the cost and reliability of the ICG and OBR replacement policies.

- A stands for the fixed costs that have to be made for each replacement action, whether preventive, corrective, large or small. For example, rental of basic equipment like a truck.
- a_j is the additional cost for the preventive replacement of group j. For example, the lamps, and extra man hours.
- c_i the additional cost for the corrective replacement of a lamp in light point i. For example, the lamp, and extra man hours.
- p_j the penalty cost for failure of any lamp in group j. This is a fictive measure of the severity of a failure in group j. It is used to steer the optimization to an 'acceptable' trade–off between efficiency and reliability.

2.2 The ICG policy

The Indirect Class Grouping (ICG) policy can now be phrased as follows.

- When a lamp fails, it is correctively replaced immediately.
- In addition, every group j is preventively replaced after fixed intervals of length $k_j T$, $k_j \in \mathbb{N}_{>0}$, $T \in \mathbb{R}_{>0}$.

We assume that failures are reported immediately! It is assumed too that the k_j and T can always be chosen by the optimization procedure such that there is at least one group ℓ for which the length of the interval $k_\ell T$ is the greatest common divisor of the others.

An example of these parameters is given in Figure 1. Here one has three groups, say $r$ed, $g$reen, and $y$ellow; T is 6 months, and $k_r = 1$, $k_g = 2$, $k_y = 4$.

The cost function for ICG for a partitioning into m groups is displayed in (1).

$$C_{ICG}(T, \vec{k}) = \frac{A}{T} + \sum_{j=1}^{m} \frac{a_j}{k_j T} + \sum_{j=1}^{m} \frac{\sum_{i \in S_j} M_j(k_j T)(A + c_i + p_j)}{k_j T} \tag{1}$$

The first two summands express the expected costs per unit of time for all groups for preventive maintenance. The last summand represents the expected costs made for corrective maintenance.

- As there is a group with $k_j = 1$, and A is fixed regardless the size of the action, A has to paid only once per T.

- The effect of the a_j depends on the k_j, e.g., if $k_j = 3$, then per interval T the additional costs for the preventive replacement of group j sum up to $a_j/3$.
- A corrective replacement of a lamp from group j costs $A + c_i + p_j$. $M_j(k_jT)$ is the expected number of corrective replacements in the interval $[0, k_jT]$ for any light point in group j. In other words, over the interval $[0, k_jT]$, corrective replacement is expected to occur $M_j(k_jT)$ times. For all lamps in group j this amounts to $\sum_{i \in S_j} M_j(k_jT)(A + c_i + p_j)$, where S_j is the index set of the lamps of group j. This sum has to be divided by the length of the interval, i.e., k_jT. And to be calculated for all groups, this results in the sum $\sum_{j=1}^{m} \cdots$.

Note that with the penalty, the reliability of the system can be controlled: an increase of the penalties results in an increase of preventive maintenance, and therefore a decrease of corrective maintenance. The optimization problem for an ICG policy is as follows:

> Given a partitioning into groups and a value for the replacement parameters, find a $T \geq 0$ and $k_j \in \mathbb{N}$ for all groups j such for which the cost function given in equation (1) has a minimal value.

van der Duyn Schouten et al. (1996, page 19) give an algorithm that, under certain assumptions, solves this problem. In the following the set of k_j's is denoted with $\vec{k}$.

The **reliability** of a policy is the average number of corrective replacement actions per time unit. The reliability for ICG is expressed in (2).

$$R_{ICG}(T, \vec{k}) = \frac{\sum_{j=1}^{m} n_j M_j(k_jT)}{k_jT} \tag{2}$$

Again, $M_j(k_jT)$ is the expected number of corrective replacements over an interval of length k_jT for a lamp from group j. When multiplied by n_j, the number of lamps in group j, we arrive at the expected replacement for the group. Divided by k_jT, this gives mean number of corrective replacements per unit of time. Finally, these figures are summed over all groups.

Note that the cost and reliability of the ICG replacement policy depends upon the partitioning of the light points into groups: different partitionings give different cost and reliability functions. The same observation holds for OBR, discussed next.

2.3 The OBR policy

The OBR policy is can be stated as follows.

- When a lamp fails, it is correctively replaced immediately.
- Every group j is preventively replaced when the time passed since the last preventive replacement is greater than a certain fixed lower bound t_j, and an **opportunity** for replacement occurs. An opportunity for replacement occurs when there is a failure of some lamp. This may be a lamp in group j but it may also be a failure of a lamp from another group (from the same TLI).

Note that with OBR, a preventive replacement is triggered by a failure. Since the occurrence of failures is not known in advance, this means that contrary to ICG, the date of the preventive replacement is not known in advance.

To specify the cost function for OBR, we need two more notions:

- The stochastic variable V_j stands for the duration after t_j for a replacement opportunity, i.e., a failure to occur in j itself.

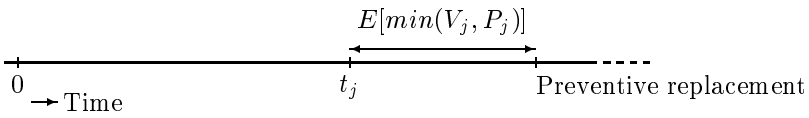

Figure 2: Time between preventive replacements of group j.

- The stochastic variable P_j stands for the duration after t_j for a replacement opportunity in another group.

The cost function for OBR is displayed in (3).

$$C_{OBR}(\vec{t}) = \sum_{j=1}^{m} \left(\frac{Pr(P_j \geq V_j)(A + a_j + p_j)}{t_j + E[min(V_j, P_j)]} + \frac{Pr(P_j < V_j) a_j}{t_j + E[min(V_j, P_j)]} + \frac{\sum_{i \in S_j} M_j(t_j)(A + c_i + p_j)}{t_j + E[min(V_j, P_j)]} \right) \quad (3)$$

As for ICG, the costs are calculated over the interval between two preventive replacements. Contrary to ICG, the length of this interval is not fixed. We only know it is at least t_j. The first two summands in the expression for the cost function represent the cost for preventive maintenance, the last summand represents the cost for corrective maintenance.

- $Pr(P_j \geq V_j)$ is the probability that a lamp fails in group j before a lamp fails in another group, i.e., the chance that $t_j + V_j$ is less or equal to $t_j + P_j$. In this case the cost of the preventive replacement is $A + a_j + p_j$. $E[min(V_j, P_j)]$ is the Expected minimal time after t_j that it takes a lamp to fail in j or in another group, i.e., the expected duration after t_j for a replacement opportunity to occur (see Figure 2).
- The second summand represents the cost of a preventive replacement of group j if an opportunity arises in another group. In this case, the fixed costs are already incurred for the other group and there is no penalty (because there is as yet no defect in group j).
- The last summand again represents the costs for a corrective replacement action.

The optimization problem is as follows:

> Given a partitioning into groups and a value for the replacement parameters, find t_j for all groups j for which the cost function given in equation (3) has a minimal value.

In the following the solution for the t_j's will be denoted with $\vec{t}$. van der Duyn Schouten et al. (1996, page 25) give an algorithm that, under certain assumptions, solves this problem. One of these is that when optimizing t_j for group j, the failures in the other groups can be modeled with a Poission process with intensity π_j. This π_j is also a result of optimization, it has however no importance from an 'external function' point of view: once the t_j's are found and fixed one forgets about the π_j's. Therefore the π_j are not represented in the specifications that follow. The reliability of the OBR policy is displayed in (4) en (5).

$$R_{OBR}(\underline{t}) = \sum_{j=1}^{m} \lambda_j(t_j) \quad (4)$$

$$\lambda_j(t_j) = \frac{n_j M_j(t_j) + 1 - \pi_j L_j(t_j)}{t_j + L_j(t_j)}. \quad (5)$$

$\lambda_j(t_j)$ is an expression for the expected mean number of failures per unit of time for group j. An explanation of this function and its constituents is out of scope for this study.

3 External Interactions

3.1 Purpose

In the rest of this report, we call the system under development *DSS* (for Decision Support System).

> The purpose of the DSS is to help control both the efficiency and the reliability of the maintenance of traffic light installations (TLIs).

Responsibilities:

- The system registers information about failures and replacements of light bulbs in TLIs.
- Upon request, the system recommends an "optimal" light bulb replacement plan, where "optimal" means that the plan is efficient and reliable with respect to certain settings.
- The system should at least be able to produce replacement plans that use the ICG and OBR replacement policies, but may also be able to generate plans that use other replacement policies.

Exclusions:

- The system shall not support the maintenance of other components of TLIs than the light bulbs.

3.2 Context

The DSS interacts with entities in its environment, which may be people, software or hardware; these are jointly called **external entities**. These external entities themselves also interact with entities. We include in the **total environment** the DSS, its external entities, and any further entities needed to understand the purpose of the system. Usually, the total environment is a goal-directed system, i.e., a system that contains a regulatory control loop.

Figure 3 shows a context diagram of the total environment of the DSS. It shows a system whose goal it is to keep the light bulbs in TLIs in working order. The rectangles represent system types and the arrows represent possible interactions. The direction of an arrow represents the direction of causality, there an instance of the system type at the tail causes the interaction. To figure out what causes what, it helps to remember that a cause must always precede the effect in time. All interactions are physical processes but some consist of the exchange of symbolic matter that has a meaning. These are called **symbolic interactions**. During a symbolic interaction, physical symbols are passed from sender to receiver. All interactions in the context diagram, except those with the TLI, are symbolic.

1. An observer observes that a light point is broken. The observer may be a member of the public, police, maintenance personnel, etc. It may also be a network that connects a TLI with a maintenance office. The observation itself is a physical process in which the cause-effect chain runs from the TLI to the observer. The observer transforms this into a symbolic process.
2. The observer notifies User 1,

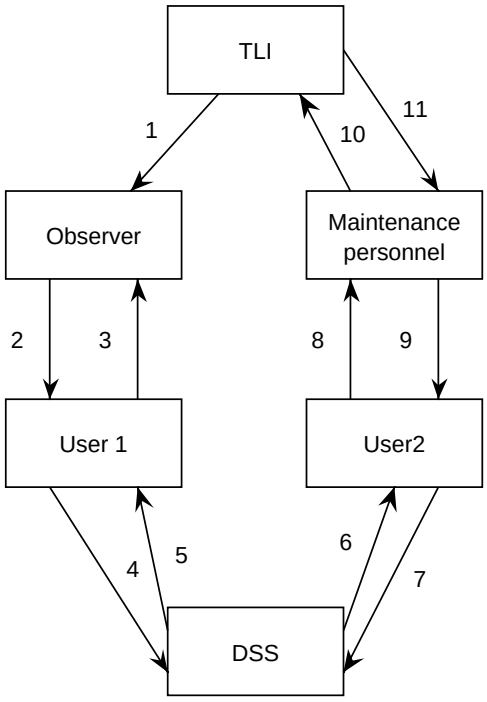

Figure 3: Context diagram.

3. who may ask for more information. They use speech, written notes or perhaps even diagrams.
4. User 1 enters the data,
5. which presumably is confirmed by the DSS.
6. The DSS notifies User 2 of replacement actions that are due,
7. which action presumably requires some activity (prompting, entering a password, etc.) of User 2.
8. User 2 passes this on to maintenance,
9. which we assume will not remain completely silent during this communication.
10. Maintenance performs the replacement actions and
11. may make some sundry observations of traffic lights.

This is the basic maintenance cycle, which gives us a clue about the desired functions of the DSS as well as of the context in which it will be used.

User 1 and User 2 are (employees of) the owner of the TLIs or of a company to which maintenance is outsourced. At the current state of analysis, it is too early to say which functionaries these are and what work procedures they should follow. Another issue to be determined later is the precise form and meaning of the data exchanged in the communications and performance issues such as frequency of data traffic, response time, amount of data, etc.

3.3 Subject Domain

Each system interacts with its external environment. If the system has symbolic interactions, then these interactions are *about* something. The symbols exchanged between the system and its environment have a meaning. The **subject domain** of a system is the part of the world in which these symbols are interpreted.

Figure 4 shows a representation of the subject domain of the DSS. On the one hand, this model cannot be made without first listing the external interactions of the DSS. On the other hand, these external interactions cannot be understood without first modeling the subject domain. In practice, the external interactions and the subject domain are modeled jointly. Here, we first present the subject domain and then in section 3.4 present the desired external interactions of the DSS.

A rectangle represents an object class, where an object is an observable entity that may or may not exist and has a unique identity. Objects of a class are called instances of the class. The rectangle may be divided into three sub-areas, which contain, from top to bottom, the class name, the names of attributes of the class instances, and the names of possible events that may occur in the life of class instances. The attributes of an instance hold the state of the class instances. Each object class has a distinguished Boolean-valued attributed called *Exists* that in each possible state of the domain is *true* for precisely the instances that exist in that state.

Arrows represent mathematical functions on object classes. The inverse of a function $f : C_1 \rightarrow C_2$ is a function $f^{-1} : C_2 \rightarrow \wp(C_1)$. A numeric constraint at the tail of an arrow represents a constraint on the cardinality of the set images of f^{-1}. Bidirectional arrows represent bijections.

Figure 4 says that to each light point there belongs exactly one bulb type and each bulb type has exactly one manufacturer. A light point is located in a traffic light, which belongs to a Traffic Regulation System (TRS). There is a one-one correspondence between TRSs and controllers. So each controller defines a TRS. Each light point accepts a certain bulb type, whicb has a lifetime distribution and a manufacturer.

Figure 5 shows the desired situation in the subject domain, that must be in place if we are going to use a DSS for TLI maintenance. In this situation, light points are classified into *groups* and a set of one or more groups is called a Traffic Light Installation (TLI). The independence from the location in TRSs means that groups, and hence TLIs, can span across several TRSs. In fact, the location of a light point in a TRS is not relevant for our optimization policies and we can expect that this information will not be represented in the DSS. Remember that figure 5 represents a decomposition of the subject domain and not of the DSS.

Each TLI has a policy, which at the time of writing is ICG, OBR or Laissez-faire. A policy has a name and may have 1 or more parameters. For example, the ICG policy has parameter T, the time interval used to compute preventive maintenance periods. OBR has no policy parameters. See section 2 for a short definition of these policies.

Because a group is assigned to a TLI and a TLI has a policy, each group can be classified as an "ICG group", an "OBR group", etc. Groups have parameters. These are explained in the dictionary of the subject domain, given below in appendix A. Each group also has a number of data that characterize the lifetime distribution of its light bulbs. All light bulbs in the group are assumed to have the same lifetime — this is the rationale for partitioning the light bulbs in groups in the first place. Jointly, the lifetime distribution data and the group parameters provide the data needed to compute the cost functions (1) and (3) given on pages 4 and 6.

The Policy and Group parameters have been modeled this way to allow addition and removal of policies without changing the subject domain model. The alternative would have been to explicitly define policy classed ICG and OBR, and define group subtypes like ICG-group and OBR-group. These would have their parameters listed explicitly as attributes. The resulting model (figure 7) is simpler to read but inflexible with regards to the addition and removal of policies. (The dashed circle means that a group can move from ICG to OBR group and vice versa.)

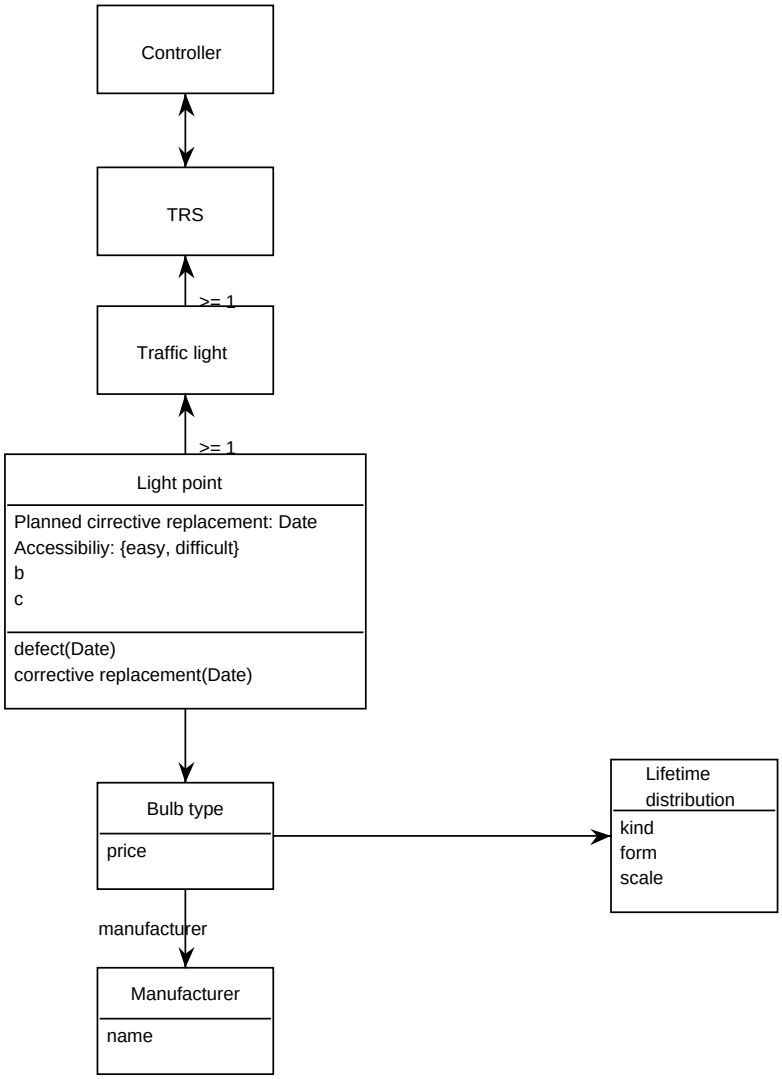

Figure 4: Subject domain decomposition as it is found before introducing maintenance decision support.

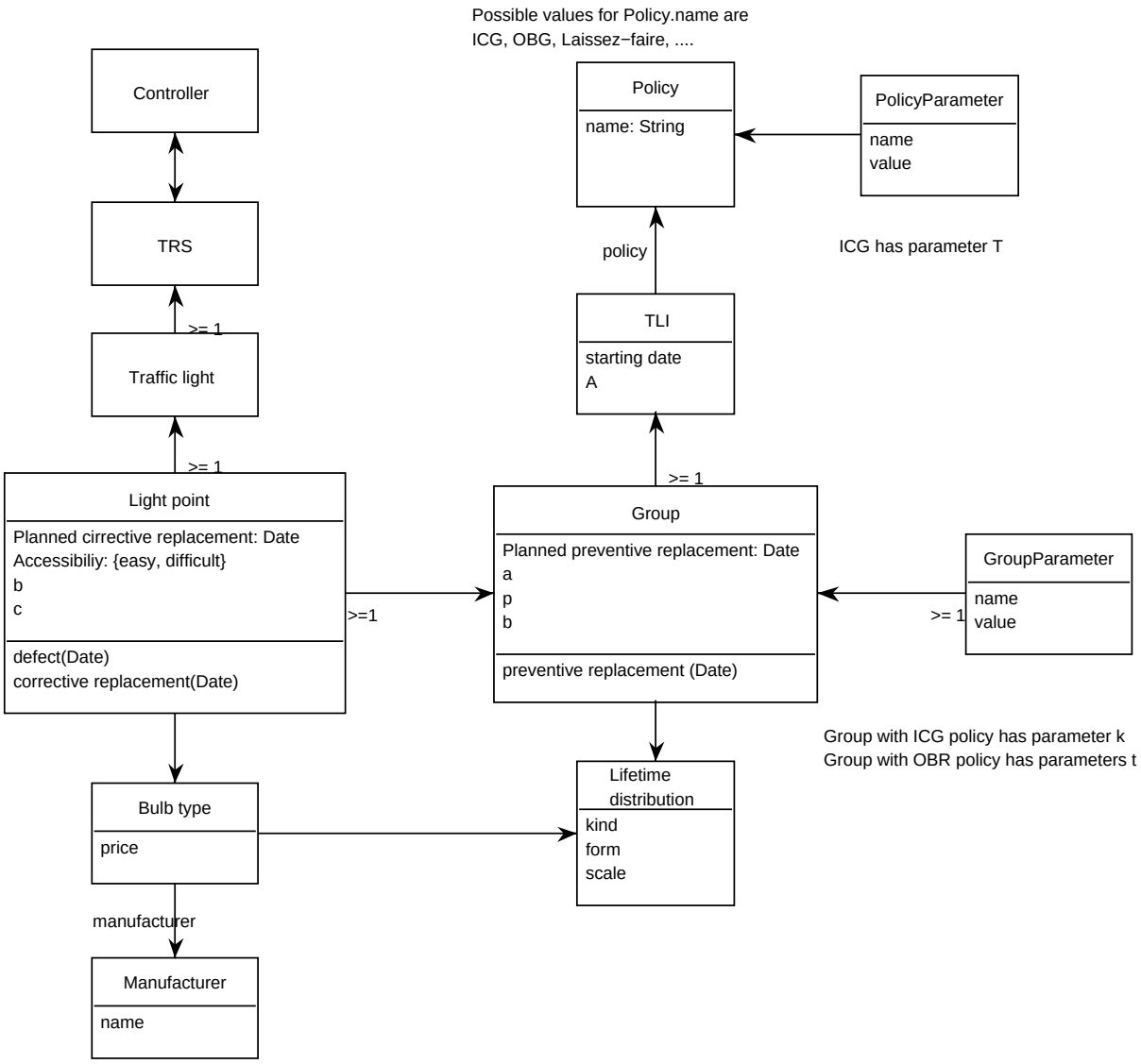

Figure 5: Subject domain decomposition after the introduction of maintenance decision support.

- The burning fraction b of a Light_point is less than or equal to the burning fraction b of its Group.

Figure 6: Integrity constraints for the subject domain components.

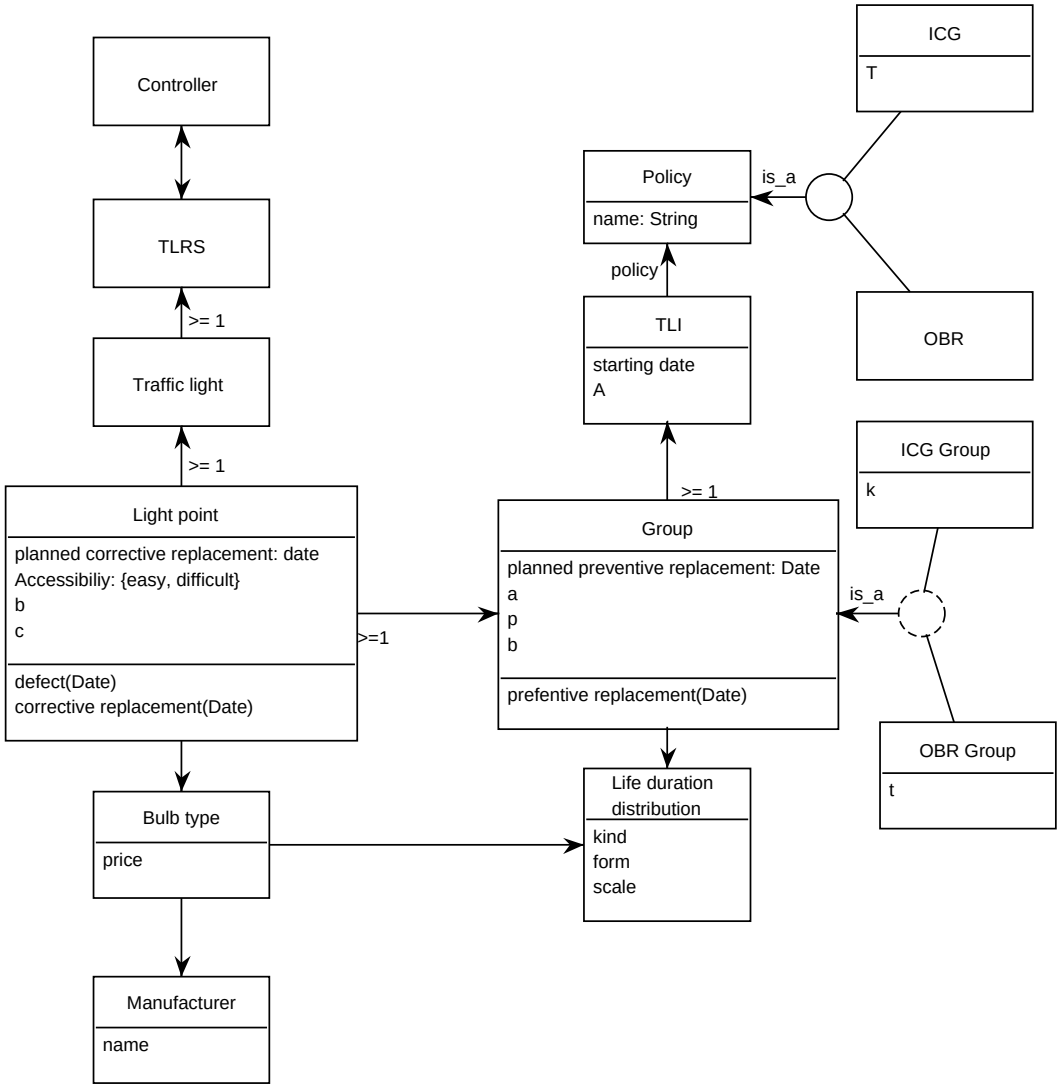

Figure 7: Model of subject domain decomposition that is easier to read but is less flexible.

Yet another solution is to list all possible parameters of all policies explicitly as Group attributes. So in figure 5, parameters k (needed for ICG) and t (needed for OBR) would be added. This has the same disadvantages as figure 7. In addition, it violates the principle that the attributes listed in a class box are applicable to instances of the class. In implementation terms, it introduces null values.

We should repeat that figure 5 models a part of the (desired situation in the) real world. Its class instances are not software objects, they are objects in the real world that exist independently from the DSS — in fact, they exist independently from any piece of software. However, the subject domain model is used in section 4 to define a conceptual decomposition of the DSS into software objects. Some of these objects correspond to objects in the subject domain.

Usually, there are some propositions appended to a subject domain model, that represent *necessary truths*, that cannot be violated by the subject domain, or *norms*, to which the subject domain must conform. Figure 6 shows a single proposition, which expresses a norm for the values of the burning fraction b of a light point and its group.

A class-relationship diagram (CRD) of a subject domain model should always be supplemented by a dictionary that defines the classes, attributes and events declared by the CRD to exist. Appendix A defines these terms informally, but as clearly as possible. A formal specification of these terms is not appropriate, because we use the dictionary to reach agreement among all stakeholders, even those who do not speak a formal language. Even stronger, without this informal dictionary, the formal specification would not have any meaning.

The function of the subject domain model with respect to the system specification is to act as a structure in which to interpret the definitions of the desired system functions and of the key terms in the specifications of these functions. We observed in section 3.2 that the interactions between the DSS and its environment are symbolic. In other words, during these interactions, symbols are exchanged with the environent. Symbols are entities which by convention are given a meaning, i.e. they refer to something outside themselves. The subject domain is defined as the part of the world to which these symbols refer.

The subject domain has the same role with respect to a formal specification as it has with respect to a semiformal specification: In a formal specification, the subject domain model as well as the desired system functions are specified in a formal language. The subject domain model then provides the mathematical structure in which these specifications are given a meaning.

3.4 Functions

A **function refinement tree** refines the mission of the DSS (its purpose) until elementary external functions are reached. The tree shows *why* each external function is present. If an external function cannot be related to the mission of the DSS, then the DSS need not have this function. Figure 8 gives a function refinement tree of the system.

The elementary functions are all symbolic interactions that are about the subject domain. We therefore specify them in terms of their effect on the subject domain. Most key terms in the function specifications are already defined in the subject domain dictionary, but there are some extra terms that do not correspond to classes, attributes or events in the subject domain. Examples are “replacement plan” and “replacement schema”. These too are defined in the dictionary of appendix A.

Note that the below functions are specified by means of pre/postcondition pairs. This is due to the fact that this is a data-intensive application. It is not necessary to specify explicitly which event triggers a function and which response should be sent to the external environment. In a control-intensive application, by contrast, event/response pairs would be important. In an application that is both data– and control-intensive, pre/postcondition pairs as well as event/response pairs would occur.

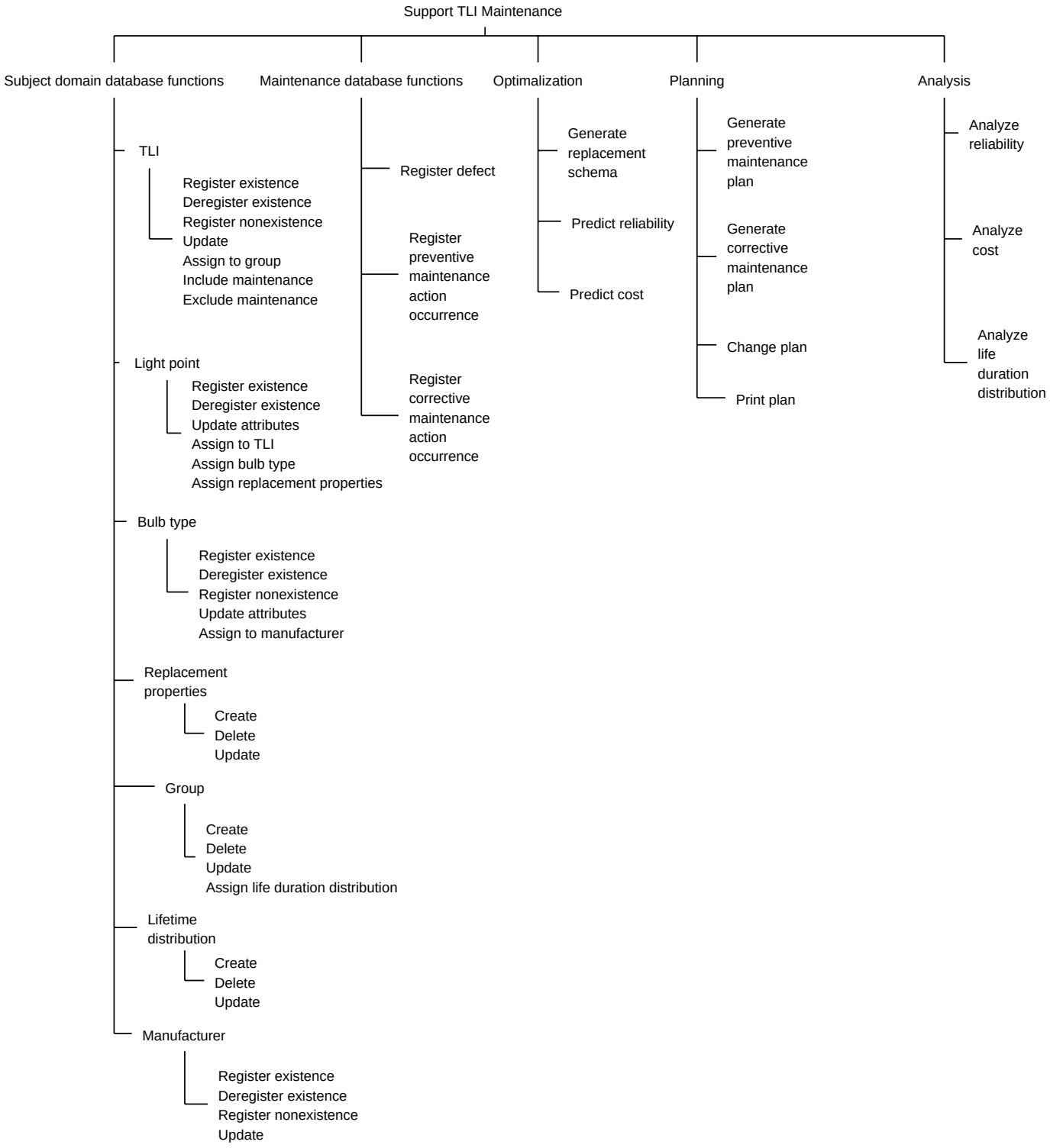

Figure 8: Function refinement tree.

3.4.1 Subject Domain Database Functions

- TLI
  - Register the existence of a TLI
    - Meaning: The existence of a TLI is registered by the DSS. The TLI already existed before the DSS registered it and it will not disappear because the DSS forgets about it. Like all external interactions, the registration action is shared between the DSS and its external environment.
    - Guards: The TLI does exist and this has not yet been registered by the DSS.
    - Pre/postcondition pairs:
      - Precondition 1: tli is an identifier not yet used for a TLI.
      - Postcondition 1: The DSS uses tli to represent the existence of the TLI.

      We describe the postcondition in terms of the concept of representation. This can be viewed as a reference to the state of the DSS and thus as an implementation-oriented specification. However, we only refer to observable state. If the DSS represents the existence of a TLI, it will be able to answer the question whether the TLI exists. We could therefore replace the postcondition with
      - The DSS is able to answer the question whether the TLI exists.

      However, given the fact that we are specifying database functions here, and the function of a database is to represent the existence of certain facts in the subject domain, we see no harm in using the concept of representation in a specification.
  - Deregister the existence of a TLI
    - Meaning: The DSS forgets about the existence of a TLI.
    - Guards: The DSS does represent the existence of the TLI.
    - Pre/postcondition pairs:
      - Precondition 1: tli is a TLI represented to exist by the DSS.
      - Postcondition 1: The DSS does not represent the existence of tli.
  - Register the nonexistence of a TLI
    - Meaning: The DSS registers the fact that a TLI does not exist anymore.
    - Guards: A TLI has ceased to exist.
    - Pre/postcondition pairs:
      - Precondition 1: tli identifies a TLI that has ceased to exist.
      - Postcondition 1: The DSS represents the fact that tli does not exist anymore.
  - Update TLI properties
    - Meaning: This is actually a collection of functions, in which attribute values of the TLI are changed.
    - Guards: The TLI is represented to exist and is currently under maintenance.
    - Pre/postcondition pairs:
      - Precondition 1: (input values and current attribute values of a tli)
      - Postcondition 1: (tli has updated attribute values)
  - Assign to group
    - Meaning:
    - Guards: The group and TLI exist and are not yet linked.

        * Pre/postcondition pairs:
            · Precondition 1: g is a group and tli is a TLI
            · Postcondition 1: g is linked to tli.
    - Include an existing TLI into maintenance
        * Meaning:
        * Guards: The TLI is not yet under maintenance.
        * Pre/postcondition pairs:
            · Precondition 1: tli is a TLI
            · Postcondition 1: tli is represented to be under maintenance and a preventive or corrective maintenance plan (as appropriate) has been generated.
    - Exclude an existing TLI from maintenance
        * Meaning:
        * Guards: The TLI exists.
        * Pre/postcondition pairs:
            · Precondition 1: tli is a TLI
            · Postcondition 1: tli's maintenance is represented as having ceased, and no maintenance is planned.

- Light point
    - Register/deregister the existence of a light point
    - Register the nonexistence of a light point
    - Update attributes
    - Assign to group
    - Assign bulb type
    - Assign replacement properties
- Bulb type
    - Register/deregister the existence of a bulb type
    - Register the nonexistence of a bulb type
    - Update attributes
    - Assign to manufacturer
- Replacement properties
    - Create
    - Delete
    - Update attributes
- Group
    - Create
    - Delete
    - Update attributes
    - Assign lifetime distribution

- Lifetime distribution
  - Create
  - Delete
  - Update attributes
- Manufacturer
  - Register/deregister the existence of a manufacturer
  - Register the nonexistence of a manufacturer
  - Update attributes

3.4.2 Maintenance Database Functions

- Register defect
  - Meaning: The DSS registers the fact that a light bulb in a light point of a TLI is defect.
  - Guards:
  - Pre/postcondition pairs:
- Register preventive maintenance action occurrence
  - Meaning:
  - Guards:
  - Pre/postcondition pairs:
- Register corrective maintenance action occurrence
  - Meaning:
  - Guards:
  - Pre/postcondition pairs:

3.4.3 Optimalization functions

- Generate replacement schema
  - Meaning: Compute the parameters needed to determine optimal replacement schemas, using the algorithms developed by van der Duyn Schouten et al. (1996).
  - Guards:
  - Pre/postcondition pairs:
    - Precondition 1: A value for A, a grouping, and for each group, values for a, p and b and for each light point a value for c.
    - Postcondition 1:
      - For the ICG policy, a value for the base cycle T (attribute of ICG policy) and for each group a value for the parameter k.
      - For OBR, a value for the parameter t for each group.
- Predict reliability

- Meaning: Compute the estimated reliability according to equations (2) (ICG) or (5) (OBR) in section 2.
- Guards:
- Pre/postcondition pairs:
  - Precondition 1: A value for A, a grouping, for each group, values for a, p, and b, for each light point a value for c and a replacement scheme (values for k and T for ICG and for t for OBR).
  - Postcondition 1: The estimated reliability according to equations (2) (ICG) or (5) (OBR) in section 2.

- Predict cost
  - Meaning: Compute estimated total cost for all groups per time unit according to cost functions of the policies (e.g. (1) (ICG) or (3) (OBR)in section 2).
  - Guards:
  - Pre/postcondition pairs:
    - Precondition 1: A value for A, a grouping, for each group, values for a, p and b, for each light point a value for c, and a replacement scheme (values for k and T for ICG and for t for OBR).
    - Postcondition 1: The estimated cost according to the cost functions of the policies (e.g. (1) (ICG) or (3) (OBR)in section 2).

3.4.4 Planning functions

- Generate preventive maintenance plan
  - Meaning: For groups with an ICG replacement schema, a preventive maintenance plan can be generated from a starting date to an end date without knowing which light bulbs are currently defect. Light bulbs are replaced for ICG groups when their allotted usage period is past. For OBR groups, a preventive maintenance plan can only be generated when it is known which bulbs are currently defect. This plan must be regenerated whenever there is a registration of a defect light bulb.
  - Guards:
  - Pre/postcondition pairs:
    - A partitioning into groups, assignment of policies to TLIs, a value of k and T for each group in an ICG TLI and of t for each group in an OBR TLI, and a starting date.
    - Output: A sequence of ⟨ date, group ⟩ pairs in ascending order of dates, that indicates which groups must be preventively replaced and what date.

- Generate corrective maintenance plan
  - Meaning: Given the current state of light bulbs (defect or not) in TLIs and a policy, a list of corrective replacements can be produced.
  - Guards:
  - Pre/postcondition pairs:
    - Input: the set of current defect light bulbs.
    - Output: a list of corrective replacement actions for defect light bulbs.

- Change plan
  - Meaning: In the interest of flexibility, plans must be changeable by human users.
  - Guards:
  - Pre/postcondition pairs:
- Print plan
  - Meaning: Merge corrective and preventive maintenance plans and then print them.
  - Guards:
  - Pre/postcondition pairs:

3.4.5 Analysis functions

- Analyze reliability
  - Meaning: Analyze reliability over a period and compare with reliability as estimated in advance.
  - Guards:
  - Pre/postcondition pairs:
    - Input: The parameters upon which a replacement schema is based and the actual list of preventive and corrective replacement actions.
    - Output: Deviation of the observed and estimated reliability and statistical significance of the deviation.
- Analyze cost
  - Meaning: Analyze cost over a period and compare with cost as estimated in advance.
  - Guards:
  - Pre/postcondition pairs:
- Analyze lifetime distribution
  - Meaning: Analyze life of light bulbs of a certain type over a period and compare with lifetime distribution given by manufacturer
  - Guards:
  - Pre/postcondition pairs:

3.5 Assumptions

A few assumptions about the environment (external entities and subject domain entities) have been made. The specification depends upon the truth of these assumptions because if an assumption is false, the specification does not specify desired properties anymore.

- We assume that all names given by all manufacturers to types of light bulbs are different.
- The lifetime distribution of a group is assumed to be the same or less "optimistic" as the lifetime distribution of the individual types of light bulbs in the group.
- Failures are reported immediately.

- The cost functions make assumptions listed by (van der Duyn Schouten et al. 1996). These do not affect the model presented here but they do affect the validity of the cost functions.
- It is assumed that the k_j and T can always be chosen by the optimization procedure such that there is at least one group j' for which the length of the interval $k_{j'}T$ is the greatest common divisor of the others.

3.6 Limitations

- The cost functions are limited by the assumptions made by van der Duyn Schouten et al. (1996).
- The replacement schemas must be regenerated when the price of light bulbs changes, or when the partitioning in groups changes.
- There is no support in the current functionality for exploring different partitionings other than trial and error.
- Also, there is no facility to revise replacement plans when corrective or preventive replacement actions did not take place, or took place at a later or earlier date than originally planned.

4 Decomposition

We now decompose the DSS into components that can jointly realize the desired external functions. This is the first step towards an implementation.

4.1 Conceptual Decomposition

A conceptual decomposition is a decomposition in terms of the external environment of the DSS. It is independent from any underlying implementation environment. This means that domain specialists are able to understand the decomposition; no knowledge of the underlying implementation environment is needed to understand the decomposition.

Figure 9 shows the conceptual decomposition of the system. It uses the same notation as that used for a subject domain decomposition, the class-relationship diagram. To make the distinction with a subject domain decomposition explicit, all names end with "-S".

The system decomposition is conceptual, because it shows a collection of conceptual object classes needed to implement the desired functions and does not deal in any way with the physical realization of these classes. The conceptual decomposition was found by defining a **surrogate class** for every subject domain class about which the system must remember data. This gives us the classes inside the TLI subject area of figure 9. In addition, the system must remember action occurrences in the subject domain. Each action occurrence is modeled by a class inside the subject area "Maintenance actions". Note that we removed the maintenance actions in Light_point_S and Group_S. Occurrence of these actions is represented by the creation of instances of Replacement_action_occurrence_S. The defect action of Light_point_S has been replaced by a defect attribute, that, if it has a value, holds the most recently registered defect occurrence date. The intention is that this attribute has a value only when a defect has occurred that has not yet been replaced.

The components of a conceptual decomposition are software objects, so the classes shown in figure 9 are classes of software objects. The instances of these classes are software objects. (In an implementation, more classes would appear, because there are likely to be details to be dealt with that are due to the facilities present or absent in the programming language.) All software objects modeled in figure 9 are for subject domain entities or for real-world event occurrences.

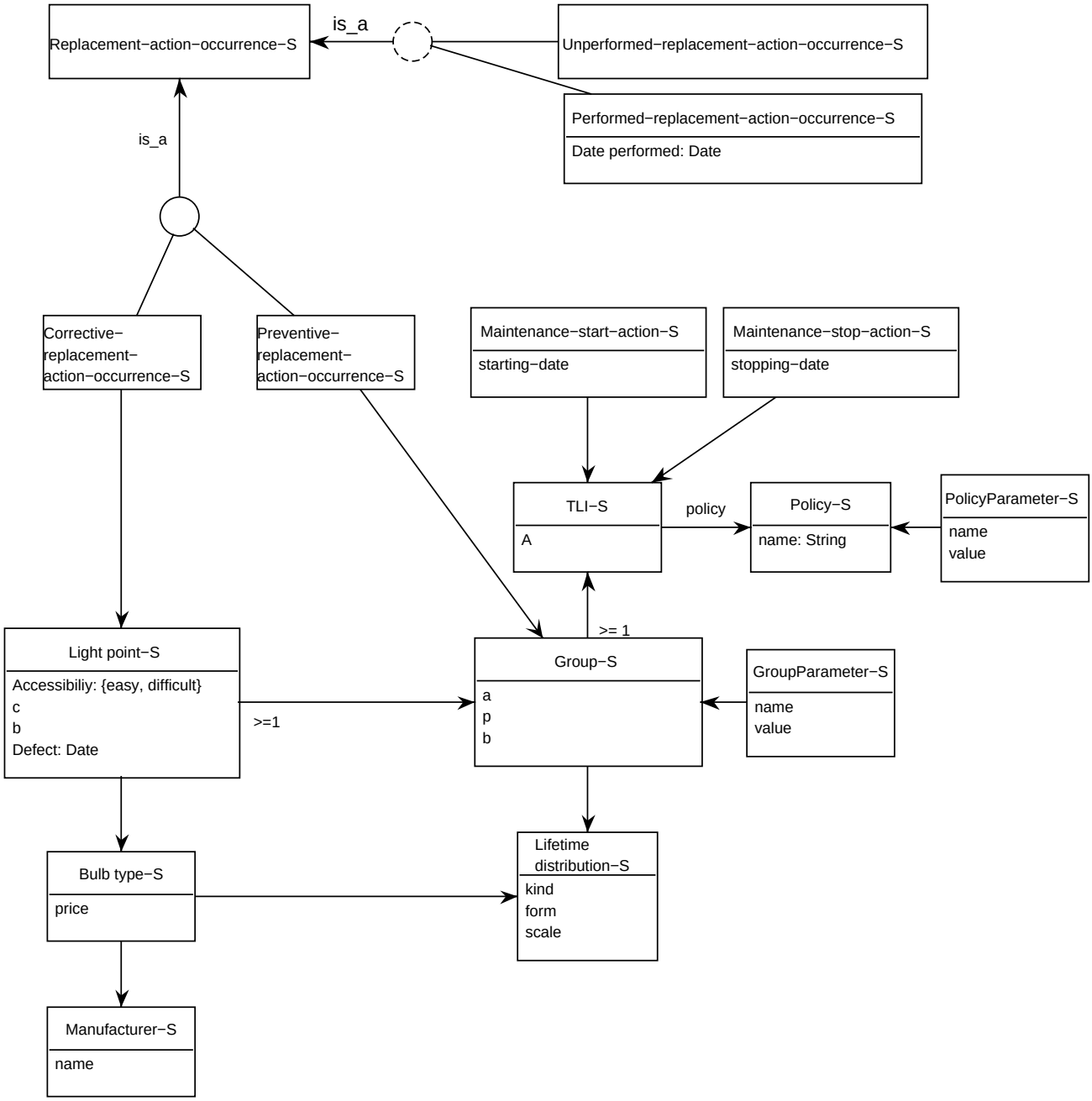

Figure 9: Conceptual decomposition of the software.

- The burning fraction b of a Light_point_S is less than or equal to the burning fraction b of its Group_S.

Figure 10: Integrity constraints for the software components.

There is one constraint applicable to the software decomposition, not shown in the decomposition diagram of figure 9. It is shown in figure 10. This is part of the specification, i.e. just as the graphical part of the specification, it is a norm to which the implementation must conform. An implementation that violates constraints, does not represent the real world. The constraint of figure 10 corresponds to the subject domain constraint listed in figure 6. A subject domain constraint can be translated in a "soft" constraint for the implementation, which is a constraint of which violations are tolerated. For example, a bank account must be non-negative, but violations of this subject domain norm are always represented by a bank account database. Alternatively, a subject domain constraint can be translated in a hard constraint for the implementation, which means that violations of the constraint by the implementation are not tolerated. This is the case for the burning fraction constraint of figure 6. By not being able to represent any violations, the implementation helps to enforce the subject domain constraint.

4.2 Allocation and Flowdown

To check if the conceptual decomposition is complete, we next allocate and flow down the desired functions to the software objects.

The tables in figures 11 and 12 show the allocation of flowdown of external functions to conceptual components by means of **function decomposition tables**. To save space, only the functions Optimization and Planning are shown here. The database functions are allocated trivially to software objects. The Analysis functions have not yet been analyzed sufficiently to allocate them to components.

The top row of a function decomposition table lists all the external functions, the leftmost column of the table lists all (classes of) conceptual components, and the entries of the table lists the local actions by which individual components participate in the external function. The meaning is that one or more instances of the component classes cooperate by means of these actions to realize the external function.

In the function predict cost (OBR), the value n needed in (3) can be computed by counting the number of instances in a group. The expected number of corrective replacements M is estimated from the lifetime distribution and burning fraction b. The same holds for the ICG reliability, equation (2). The probability distributions P and V and the intensity π needed for the OBR cost and reliability are estimated from the lifetime distribution.

5 Discussion and Conclusions

5.1 Additional properties

The function refinement tree and function specifications only give a subset of the properties we want to specify. Other properties include properties that are hard to operationalize, such as speed, capacity and user-friendliness, and constraints such as the platform on which the DSS must run, interoperability constraints, etc.

Also, there are a number of functions that may be added to those already specified, such as the following:

- Exploration of possible groupings with a view to optimizing maintainability.
- Maintaining a history of groupings, of maintenance plans, and in general, a history of all updates.
- Consistency monitoring

	Generate replacement schema (ICG)	Generate replacement schema (OBR)	Predict reliability (ICG)	Predict reliability (OBR)	Predict cost (ICG)	Predict cost (OBR)
TLI−S	Read A	Read A	Read A	Read A	Read A	Read A
Policy−S						
PolicyParameter−S	Write T		Read T		Read T	
Light point−S	Read c	Read c	Read c	Read c	Read c	Read c
Bulb type S						
Manufacturer−S						
Group−S	Read a Read p Read b	Read a Read p Read b	Read a Read p Read b	Read a Read p Read b	Read a Read p Read b	Read a Read p Read b
GroupParameter−S	Write k	Write t	Read k	Read t	Read k	Read t
Lifetime distribution−S	Read kind Read form Read scale	Read kind Read form Read scale	Read kind Read form Read scale	Read kind Read form Read scale	Read kind Read form Read scale	Read kind Read form Read scale
Replacement−action−occurrence−S						
Performed−replacement− action−occurrence−S						
Unperformed −replacement − action−occurrence−S						
Corrective−replacement− action−occurrence−S						
Preventive−replacement− action−occurrence−S						
Maintenance−start−action−S						
Maintenance−stop−action−S						

Figure 11: Decomposition of optimization functions.

	Generate preventive maintenance plan (ICG)	Generate preventive maintenance plan (OBR)	Generate corrective maiontenance plan	Change plan	Print plan
TLI−S					
Policy−S					
PolicyParameter−S	Read T				
Light point−S		Read defect	Read defect		
Bulb type S					
Manufacturer−S					
Group−S					
GroupParameter−S	Read k	Read t			
Lifetime distribution−S					
Replacement−action−occurrence−S	Write	Read Write			
Performed−replacement− action−occurrence−S		Read			
Unperformed −replacement − action−occurrence−S	Write	Write	Write		
Corrective−replacement− action−occurrence−S			Write		
Preventive−replacement− action−occurrence−S	Write	Read Write			
Maintenance−start−action−S	Read				
Maintenance−stop−action−S	Read				

Figure 12: Decomposition of planning functions.

5.2 Further decomposition

The conceptual decomposition can be refined by adding components that implement function objects and interface objects and interfaces.

- Functions can be implemented by function objects. For example, a PredictCost object could send the appropriate messages to other objects in order to compute the cost. However, this really is a mathematical function and an alternative is therefore to implement an Optimization class with methods ComputeParameters, Predictreliability and PredictCost.
- Interface objects could be added to deal with user interfaces and interfaces to other devices.

To move to a physical decomposition, we should model the hardware configuration architecture upon which the conceptual objects must run. This includes the topology of processing nodes and connections, the people with whom the system interacts, and an allocation and flowdown of conceptual objects to physical components.

5.3 Formal specification

In a logic-based specification, a class-relationship diagram (CRD) such as shown in figures 5 and 9 corresponds to an order-sorted signature in first-order logic. For example, the *Light_point_S* and *Group—S* classes can be declared as follows. We suppose an enumeration sort $Access = \{easy, difficult\}$ and sorts *Natural*, *Rational* and *Money* have been defined. The formula $[\alpha]\phi$ stands for "after action α, formula ϕ is true".

sort
  Light_Point_S
updatable functions
  accessibility : Light_Point_S → Access
  c : Light_Point_S → Money
  b : Light_Point_S → Rational
  defect : Light_Point_S → Date (partial)
nonupdatable functions
  set_accessibility : Light_Point_S × Access → Event
  set_c : Light_Point_S × Money → Event
  set_b : Light_Point_S × Rational → Event
  set_defect : Light_Point_S × Date → Event
axioms
  $\forall$ *l : Light_point_S, a : Access :: [set_accessibility(l, a)] accessibility(l) = a*
etc.

sort
  Group_S
updatable functions
  a : Group_S → Money
  p : Group_S → Natural
  b : Group_S → Rational
nonupdatable functions
  set_a : Group_S × Money → Event
  set_p : Group_S × Natural → Event
  set_b : Group_S × Rational → Event
axioms

∀ *g: Group_S, m: Money :: [set_a(g, m)] a(g) = m*

etc.

Global constraints

∀ *g: Group_S :: ∃ l : Light_point_S:: group_s(l) = g*

∀ *g : Group_S , l : Light_point_S :: group_s(l) = g → b(g) ≥ b(l)*

The precise semantics of this is defined elsewhere (Wieringa et al. 1995). Basically, the sorts *Light_point_S*, Group_S etc. are interpreted as sorts in an abstract data type, that in an initial algebra semantics contain only identifiers, i.e. constants of the form $next^n(e_0)$ for a natural number n and a generator element e_0. The abstract data types (*Natural*, *Rational* and *Money* and the identifier sorts) jointly define the domain of a labeled transition system. Each state in this system has the same domain, which consists of the abstract data types. The only thing that may differ across the states is the intepretation of attribute symbols. The edges in the transition system are labeled by actions, that may change the attributes according to the constraints plus some minimal semantics that we summarize by the slogan "minimal change, maximal reachability".

The specification on which the study in this report is based was written in PSF. PSF combines the process algebra ACP (Baeten & Weijland 1990) with the algebraic data type formalism ASF (Bergstra et al. 1989). This specification of the DSS system contained a specificitation in ASF of abstract data types as mentioned above and dynamic parts in process algebra that corresponds to the transition system mentioned above. Data terms that form the arguments of the process terms can be looked upon as attributes that change. The difference between the two approaches is that the TRADE specification is directed at fixing the concepts, the external functions of the system in relations to these concepts, and the boundary of the state space, whereas the PSF specification is directed at generating the state space, and the precise properties of the concepts in an operational style. In other words, TRADE is suited for requirements engineering, PSF for system decomposition.

This suggests that the two modeling techniques could be used together in product development, as subsequent means for modeling. However, we did not fully investigate the technical consequences.

If we choose a formalization in pure process algebra, we may define for each software object class an action that creates a process that is an instance of that class, with a local state consisting of the attributes of the class. The instance has read actions that allow other processes to read its state, and write actions that allow other processes to update its state. Some instances also have update actions, e.g. *Light_Point_S*, which has local actions *defect(Date)*, *corrective_replacement(Date)* and *preventive_replacement*.

Next, each external function must be specified as a process that communicates with a number of instances in order to compute the desired output.

The classical reason for formalization is to increase confidence in the specification or its implementation.

- The requirements specification can be formalized to make sure that no important aspect has been forgotten. Usually, during formalization, one stumbles upon aspects that were forgotten when writing the informal or semiformal specification. Completeness is enhanced when the model can be *executed*. In this case, the PSF specification could be interpreted to see whether it works in the desired fashion. The disadvantage of model execution is that one is also forced to add information that is not relevant for the specified requirements but that is needed to execute the model. Moreover, in the current example, the non-executable semiformal model probably is sufficient to make us belief that the system, when implemented, does indeed behave as desired.

- A second way to increase confidence in the specification is by means of *prototype code generation*. The PSF specification is compiled into a programming language. This could be done to get a quick and dirty system version that can be given to the users, for example to test the user

interface or some other aspect of external behavior. Again, in this case there is probably little need to do this.

- Formalization can also be used for correctness proofs. Assuming the specification is valid, i.e. it specifies what is desired, we can make sure that the implementation does what the specification says by means of *correctness proofs*. This may be desirable in the case of safety-critical applications, but the DSS is not one of them.

- Correctness can also be obtained by means of *correctness-preserving transformations*. This is a misnomer, for what is meant is that the transformations preserve the meaning of the specification, and therefore produce correct code. Again, for the DSS this is not really needed.

A different reason for formalization is to create a library of models of a certain domain, that can be used to quickly develop products that need these domain models. This argument is applicable to semiformal as well as formal models. For example, the domain of traffic regulation systems can be modeled by means of diagrams, as done in this report, and these models can be used to speed the development of IT-related products such as traffic light controllers, traffic monitoring systems, maintenance decision support systems, etc.

The reader may wonder, after this analysis, why the PSF specification was written anyway. It seemed clearly not the most sensible thing to do. This is partly true. In the integrated project reported on in (van Vlijmen 1998) to apply algebraic specification means this was one of the domains that simply had to be covered too.

A Subject Domain and System Function Dictionary

Any subject domain model must be supplemented with informal but precise definitions of the modeled classes, attributes and events. These are the key terms that appear in the function specifications. There are some additional key terms in the function specifications that also must be defined. Informal and clear definitions should take care that these terms are understood in the same way by the customer, specifyer and implementor of the system. The below definitions must be read with the subject domain model and global functionality of the system in mind.

In each definition, the definiendum is written in **boldface**. Words in the definiens that are written in *italic* are defined in the dictionary itself, i.e. they occur elsewhere in the dictionary as definiendum.

- **Base cycle**: A time interval used by the ICG policies for computing preventive maintenance intervals.

- **Bulb type**: A kind of light bulb, produced by one manufacturer. The manufacturer's classification is used.

  - **Name**: Name of the bulb type. The manufacturer's name for this type is used. We assume that all names given by all manufacturers are different.
  - **Price**: The price at which the bulb must be bought.

- **Controller**: a system that controls traffic by means of a set of *traffic lights*.

- **Efficiency** of a replacement policy: Cost of the policy per unit time. The cost functions of the *ICG* and *OBR* policies are given in formulas 1 and (3 in section 2. The efficiency of a policy depends upon the partitioning of *light points* into *group*s and of groups into *TLI*s, as well as upon the group parameters and the replacement cost c. Given these data, the values of k for each group, t, T, P and V and the efficiency and reliability can be computed.

- **Group**: a collection of *light points* that are treated as a whole by the replacement policy. In particular, the lifetime distribution and burning fraction of all light bulbs in a group is assumed to be the same. Regardless of the policy followed for the group, the following parameters characterize a group. These cover most of the variables used in the cost formulas (1) and (3) defined in section 2.
  - **a**: Additional cost of a replacement action in which all bulbs in the group are replaced. This includes material costs (bulbs), personnel (person hours worked on this group) etc.
  - **p**: This is number that represents the undesirability of the failure of one light bulb in the group.
  - **b**: The "burning fraction", which is the fraction of the usage time that the light bulbs in this group burn.
  - **Planned preventive replacement**: Next date at which a replacement of all all light bulbs in this group is planned to take place.
  - **Preventive replacement(Date)**: an action that takes place at a date, in which a all bulbs in the group are preventively replaced.
- **GroupParameter**: A parameter that characterizes the replacement policy followed for the group. Each parameter has a **name** and a **value**. Currently, the following parameters are allowed.
  - ICG group parameters:
    - **k**: The number of *base cycles* between two successive preventive replacements of the group.
  - OBR group parameters:
    - **t**: The minimum time that must elapse between two preventive group replacements.
    - **P**: A stochastic variable that represents the time interval from t till the first opportunity for preventive replacement of the group. If this is group i, such an opportunity is defined as a failure occurrence in *another* group, j, that provides the occassion to do a replacement in the group i without incurring the basic tariff (which is attributed to j) nor the penalty cost (because there is no failure in i).
    - **V**: A stochastic variable that represents the time interval from t till the first failure occurrence in this group.
- **Lifetime distribution**: distribution function of the period of time that a *light bulb* is nondefective when it burns continuously.
  - **Kind**: Weibull or Erlang.
  - **Form**: form parameter of the distribution function.
  - **Scale**: scale parameter of the distribution.

  There is a lifetime distribution for individual bulb types and for groups. The group lifetime distribution represents the "typical" lifetime distribution of bulbs in this group. These bulbs may or may not be of different types.
- **Light point**: A socket in which a light bulb can be mounted.
  - **Accessibility**: If the light point is high, a tower wagon is needed to replace it. If it is low, no such wagon is needed.

  - **b**: The "burning fraction", which is the fraction of the usage time that the light bulb burns.
  - **c**: Additional cost of replacing one light bulb in the group. It includes material, personnel etc.
  - **Defect(Date)**: an event that occurs at the Date at which a defect is observed. (This may be later than the date at which the defect arose and earlier than the date at which it is reported.)
  - **Planned corrective replacement**: For a defect light bulb the date at which it will be replaced.
  - **Corrective replacement(Date)**: an action that takes place at a date, in which a defective bulb is replaced.

- **Reliability**: the expected average number of corrective replacements per time unit. See equations (2) and (5) in section 2.

- **Replacement schema**: If the policy is ICG, a replacement schema consists of a value of k for that group. If it is OBR, the schema consists of a value for t for that group.

- **Replacement plan**: A sequence of replacement actions, where each replacement action is planned to occur on a date. A plan has a starting date and a finishing date.

- **TLI**: *Traffic light installation*: a set of one or more *light points*.
  - **A**: basic tariff of a replacement action for this TLI. This is the minimal cost of replacing at least one bulb in the TLI. It includes personnel costs for traveling to and from the location of the group, the cost of renting and maintaining a tower wagon, etc.
  - **Starting date**: date at which maintenance for this TLI by the DSS started.

- **TRS**: *Traffic Regulation System*. A system consisting of at least one *traffic light* and exactly one *controller*.

- **Traffic light**: a collection of *light points* that jointly give instructions for road users. For example, a collection of three light points (red, yellow, green) instruct car drivers to stop or continue. For city buses and trams, there are traffic lights consisting of nine light points, arranged in a 3 x 3 matrix, that either point in the direction of permitted movement or instruct the bus or tram driver to stop. Other kinds of traffic lights exist as well.

- **Policy**: A replacement strategy. There are currently two policies, called ICG and OBR. In **ICG** (*Indirect Class grouping policy*), a light bulb is replaced immediately when it is defect. In addition, the light bulbs in each group are replaced preventively at fixed time intervals, where each group has its own time interval.

  In **OBR** (*Opportunity based replacement policy*), a light bulb is replaced immediately when it is defect. If a sufficiently long time has passed since the last time this group of traffic lights has been preventively replaced, and an opportunity for replacement arises, then all bulbs in this group are replaced.

  - **Name**: identifier of the policy.

References

Baeten, J. & Weijland, W. (1990), *Process algebra*, Vol. 18 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press.

Bergstra, J., Heering, J. & Klint, P., eds (1989), *Algebraic specification*, ACM Press Frontier Series, The ACM Press in co-operation with Addison-Wesley.

Cho, D. & Parlar, M. (1991), 'A survey of maintenance models for multi-unit systems', *European Journal of Operational Research* **51**, 1–23.

Dekker, R., van der Duyn Schouten, F. & Wildeman, R. (1997), 'A review of multi-component maintenance models with economic dependence', *Zeitschrift für Operations Research*. to appear.

Mauw, S. & Veltink, G. (1989), An introduction to PSFd, *in* J. Diaz & F. Orejas, eds, 'Proc. International Joint Conference on Theory and Practice of Software Development TAPSOFT '89', Vol. 352 of *LNCS*, Springer–Verlag, pp. 272–285.

Mauw, S. & Veltink, G. (1990), 'A process specification formalism', *Fundamenta Informaticae* **XIII**, 85–139.

Mauw, S. & Veltink, G., eds (1993), *Algebraic specification of communication protocols*, Vol. 36 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press.

van der Duyn Schouten, F., Klusener, A., van Vlijmen, S. & Vos de Wael, S. (1996), A decision support system for the maintenance of lights of traffic regulation systems, Technical Report 159, Utrecht University, Logic Group Preprint Series of the Department of Philosophy. Submitted, *IMA Journal on Maintenance Modelling*.

van Vlijmen, S. (1998), Case Studies in Industrial Application of Algebraic Specification, PhD thesis, Utrecht University. Forthcoming.

Veltink, G. (1993), 'The PSF toolkit', *Computer Networks and ISDN Systems* **25**, 875–898. Elsevier Science Publishers.

Veltink, G. (1995), Tools for PSF, PhD thesis, University of Amsterdam.

Wieringa, R. (1993), A method for building and evaluating formal specifications of object-oriented conceptual models of database systems (`MCM`), Technical Report IR-340, Faculty of Mathematics and Computer Science, Vrije Universiteit.

Wieringa, R. (1995), 'Combining static and dynamic modeling methods: a comparison of four methods', *The Computer Journal* **38**(1), 17–30.

Wieringa, R. (1996), *Requirements Engineering: Frameworks for Understanding*, Wiley.

Wieringa, R. (1997*a*), Advanced object-oriented requirement specification methods, Technical report, Faculty of Mathematics and Computer Science, *Vrije Universiteit*. Tutorial presented at the *International Symposium of Requirements Engineering*, 6–10 january 1997, Annapolis, U.S.A.

Wieringa, R. (1997*b*), A survey of structured and object-oriented software specification methods and techniques, Technical report, Faculty of Mathematics and Computer Science, *Vrije Universiteit*, De Boelelaan 1081a, 1081 HV Amsterdam. Submitted.

Wieringa, R. (1997*c*), A toolkit for requirements and design engineering (TRADE): Tool definitions and directions for use, Technical report, Faculty of Mathematics and Computer Science, *Vrije Universiteit*, De Boelelaan 1081a, 1081 HV Amsterdam.

Wieringa, R. & Saake, G. (1996), 'A formal analysis of the Shlaer-Mellor method: towards a toolkit for formal and informal requirements specification techniques', *Requirements Engineering* **1**, 106–131.

Wieringa, R., Jonge, W. d. & Spruit, P. (1995), 'Using dynamic classes and role classes to model object migration', *Theory and Practice of Object Systems* **1**(1), 61–83.