CAVIAR: A Case Study in Specification

by

Bill Flinn
Standard Telecommunication Laboratories
Harlow, England

and

Ib Holm Sørensen
Programming Research Group
Oxford University Computing Laboratory

Technical Monograph PRG-48

June 1985

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD
England

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD
England

CAVIAR: A Case Study in Specification

Bill Flinn and Ib Holm Sørensen

Abstract

This paper describes the specification, written in the specification language known as Z, of a reasonably complex software system. Important features of the Z approach which are highlighted in this paper include the interleaving of mathematical text with informal prose, the creation of parametrised specifications, and use of the Z schema calculus to construct descriptions of large systems from simpler components.

Contents

0. Introduction

This paper presents a case study in system specification. The notation used to record the system's properties is known as Z [1, 2, 3]. Z is based on set theory, and its use as a specification language has been developed at the Programming Research Group at Oxford University. Some important aspects of the Z approach are illustrated in this paper.

As is well known, software development can be divided into several phases; requirements analysis, specification, design and implementation. Z can be applied in both the specification and design phases; however, in this paper we will address the specification phase only.

We view a specification as having a two-fold purpose: firstly, to give a formal (mathematical) system description which provides a basis from which to construct a design. Such a mathematical description is essential if we are to prove formally that a design meets its specification. Secondly, to give an informal statement of the system's properties, in order that the specification can be tested (validated) against the (usually informal) statement of requirements. Thus the Z approach is to construct a specification document which consists of a judicious mix of informal prose with precise mathematical statements. The two parts of the document are complementary in that the informal text can be viewed as commentary for the formal text. It can be consulted to find out what aspects of the real world are being described and how it relates to the informally stated requirements. The formal text on the other hand provides the precise definition of the system and hence can be used to resolve any ambiguities present in the informal text. A beneficial side effect for practitioners writing such documents is that their understanding of the system in question is greatly helped by the process of constructing both the formal and the informal descriptions.

It is often the case that the process of abstraction used to construct a specification results in structures which are more general than those actually required for the system being considered. It is part of the Z approach to identify and desribe such general structures. These descriptions can be placed in a specification library. Particular cases of these general components can then be used later, either as part of the current system or in subsequent projects.

This specification case study develops a number of general systems which are subsequently constrained and combined to form the complete system description.

1. The Case Study

This specification of a Computer Aided Visitor Information And Retrieval system resulted from the analysis of a manual system concerned with recording and retrieval of data about arrangements for visitors and meetings at a large industrial site. Standard Telecommunications Laboratories (U.K.) sponsored the study in order to investigate the feasibility of converting to a computer based solution. Of particular concern were the interrelation of the stored information, the quality of the user interface and the volume of data which was required to be processed. The customer provided as input to the study an informal requirements document. We attempt to provide in this paper an outline of the steps involved in development of the eventual formal specification. It is important to stress at the outset that we view the task of constructing such a specification to be an iterative process, involving several attempts at construction of a model for the system interspersed with frequent dialogues with the customer to clarify details which are ambiguous or undefined in the initial requirements document, and frequent redrafting to clarify the structure of the document.

At an early stage in the analysis it became clear that the CAVIAR system consisted of several highly independent subsystems. Each subsystem records important relationships within the complete system and these separate subsystems are themselves related according to some simple rules. Most of the operations to be provided in the user interface can be explained as functions which transform one particular subsystem only, leaving the others invariant. These observation led to the decision to first define the subsystems in isolation and then to describe the complete system by combining the definitions of the subsystems. Once this decision had been taken, it also became clear that each of the individual subsystems, when viewed at an appropriate level of abstraction, was a particular instance of a general structure. From this vantage point it was natural to specify each of the subsystems by "refining" a specification which describes the underlying general system.

The process of analysis as presented here begins with an identification of the sets which appear to be important from the customer's point of view. Next the relationships between these sets are investigated and a preliminary classification of the subsystems follows. The third phase consists of developing an appropriate general mathematical structure in which to place these subsystems. Various ways of *specialising* (restricting) the general structure are then investigated and particular subsystems are modelled by *instantiation*. Finally the subsystem models are combined.

2. Identification of the Basic Sets

We now present a brief account of the existing system, emphasizing the important concepts in boldface. **Visitors** come to the site to attend **meetings** and/or consult Company employees. A visitor may require a **hotel reservation** and/or **transport reservation**. Each meeting is also required to take place in a designated **conference room**, at a certain **time**. A meeting may require the use of a **dining room** for lunch, on a particular date. Booking a dining room requires **lunch information** including the number of places needed. Each conference room booking requires **session information** about resources required for use in the meeting, e.g., viewgraphs, projectors. The main operations required at the user interface can briefly be described as facilities for booking, changing and cancelling the use of resources. We list below the sets together with the names that we shall adopt for referring to them.

Set	Name
Meetings	M
Visitors	V
Conference Rooms	CR
Dining Rooms	DR
Lunch Information	LI
Session Information	SI
Hotel reservation	HR
Transport reservation	TR

The informal interpretation of these sets is straight forward and for the purpose of this specification no further detail is necessary. Note that the question of modelling time remains to be resolved; at this point we simply observe that hotel reservations are made for particular *dates*, transport reservations are made for certain *times* on particular dates, and conference room bookings are made for *sessions* on particular dates. We shall not specify the term *session* further apart from noting that a date is always associated with a session; it could, for example, denote complete mornings or afternoons, or hourly or half-hourly intervals, depending on the way conference rooms are allocated.

The notion of time and the relationship between the different units of time used within the system can be formalised by asserting the existence of three sets as follows:

```
Date
Session
Time
```

together with two total functions

```
date-of-session : Session → Date
date-of-time    : Time    → Date.
```

3. The Subsystems of CAVIAR

The first approach to a mathematical model stems from the realisation that several of the sets listed above can be viewed as resources and other sets viewed as users of those resources. We can identify the following subsystems of CAVIAR in this framework (i. e., Resource-User systems). Observe that in different subsystems the same set may appear in differing roles.

System	Resources	Users
CR-M	Conference rooms	Meetings
DR-M	Dining rooms	Meetings
M-V	Meetings	Visitors
HR-V	Hotel reservations	Visitors
TR-V	Transport reservations	Visitors

Once we have made this mathematical abstraction it seems worthwhile to develop a general theory of such resource-user systems for the following reasons:

1. A specification of such a general system would be more useful as part of a "specification library" than a specific instance of such a system. Re-usability is much more likely to be achieved by having *generic* specifications available which can be *instantiated* to provide particular systems.

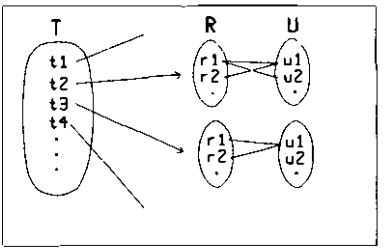
2. Particular subsystems of the general system can be *constructed* as *special cases* of the general specification in various ways. This will amply repay care and time spent on the general case. Furthermore, such instantiation may well result in a more compact implementation.

4. A General Resource-User System

We consider a system parametrised over three sets;

$$[\ T,\ R,\ U\ ]$$

Informally, T is to be thought of as a set of *time slots*, R is a set of *resources* and U is a set of *users*. We describe a general resource-user system as a function from T to the set of relations between R and U. Thus we have a rather general framework: for each time slot $t \in T$, some users are occupying or using some resources. The set T will later be instantiated with different sets in the various applications. Notice that considering *relations* between R and U allows us the possibility of a user occupying several different resources simultaneously, as is shown informally in the following diagram:



Formally, the structure we are describing is captured by a function of type

$$T \rightarrow (R \leftrightarrow U)$$

We shall now incorporate this into a *schema definition*. This schema is parametrised over the sets T, R and U, and contains some useful ancillary concepts in addition to the function ru above which will be useful in later analysis. In Z specifications it is common to introduce such derived components: as specifiers of software we are neither in the position of a pure mathematician looking for a particularly sparse set of concepts and axioms with which to define a mathematical structure, nor are we in the position of an implementor trying to minimise storage. The component *in-use*, which gives the set of resources in use at any point of time, will be useful in contexts where

we are not concerned with the user component of the system state. The function users, which gives the users occupying resources at any point of time, will be used in situations where we do not require the information about resources. We also note that there may be occasions when we wish to consider the set of *inverse* relations generated by ru; we call this function ur.

```
R-U
ru      : T → (R ↔ U)
in-use  : T → ℙ R
users   : T → ℙ U
ur      : T → (U ↔ R)
─────────────────────
∀t: T •
    in-use(t) = dom(ru(t)) ∧
    users(t)  = rng(ru(t)) ∧
    ur(t) = (ru(t))⁻¹
```

The *initial state* of this system is defined by making ru(t) the empty relation for each t.

```
Init-R-U  ≙  [ R-U | rng(ru) = { {} } ]
```

Our first theorem proves that such an inital state is reasonable and assures us of the consistency of the definition of R-U.

Theorem 1.

```
⊢ ∃ R-U • Init-R-U
```

In the interests of readability we have not given proofs of theorems stated in this paper.

We continue by defining the appropriate operations for this structure. The first step is to identify *commonalities*. For our purposes, the operations that we wish to consider on this structure are concerned with making a new booking, i. e., adding a new pair (r, u) to an existing relation at some time t, cancelling an existing booking, i. e., removing such an (r, u) pair, or modifying in some other way the relation that exists at some particular time. In fact we shall be a little more general and define a class of operations on R-U which allows the image of a *set* of time values to be altered. This is because we anticipate such operations as booking a conference room

for a meeting which lasts for several time slots. Of course a booking which involves only a single time slot is a particular case.

Thus we may summarise the common part of all the operations as follows. Their description involves: a state before, R-U which introduces ru, in-use, users and ur; a state after, R-U′ which introduces ru′, in-use′, users and ur; a set of time values, t? which denotes an input. The operations always leave the function ru unchanged except for times in t?. Formally this is captured by

```
ΔR-U
R-U
R-U′
t? : ℙ T
─────────────────
t? ⩤ ru′  =  t? ⩤ ru
```

We now have a successful *booking* operation defined as follows

```
R-U-Book
ΔR-U
r? : R
u? : U
─────────────────
∀ t : t? •
   (r?, u?) ∉ ru(t) ∧
   ru′(t) = ru(t) ∪ { (r?, u?) }
```

Thus R-U-Book inherits all the properties of ΔR-U. Furthermore, it takes two additional (input) parameters r?:R and u?:U, and is constrained by a predicate which imposes a requirement on the input parameters and also further relates the before and after states.

Notice that we are making the predicate

$$\forall\, t : t? \bullet (r?, u?) \notin ru(t)$$

a pre-condition for a successful booking. In fact, we can show that this condition is sufficient for performing a successful booking, i. e., if we are in a valid system state with the required input parameters of the correct type available and furthermore the above condition holds, then there exists a resulting valid system state which is related to the starting state according to the R-U-Book schema. Formally, this is the content of the following result:

Theorem 2

$$R\text{-}U \wedge [t? : \mathbb{P}\, T;\ r? : R;\ u? : U \mid \forall\, t : t? \bullet (r?, u?) \notin ru(t)\ ] \vdash \exists\, R\text{-}U' \bullet R\text{-}U\text{-}Book$$

A successful cancellation operation may be defined via

R-U-Cancel

$$\Delta R\text{-}U$$
$$r? : R$$
$$u? : U$$

$$\forall\, t : t? \bullet (r?, u?) \in ru(t) \wedge ru'(t) = ru(t) - \{ (r?, u?) \}$$

The pre-condition for successful cancellation is that the pair $(r?, u?)$ is related by $ru(t)$ for all time values t in $t?$; i. e., the following theorem holds.

Theorem 3

$$R\text{-}U \wedge [\ t? : \mathbb{P}\, T;\ r? : R;\ u? : U \mid \forall\, t : t? \bullet (r?, u?) \in ru(t)\ ] \vdash \exists\, R\text{-}U' \bullet R\text{-}U\text{-}Cancel$$

So far we have only specified successful operations; thus these descriptions are incomplete. We could at this stage define robust operations by introducing appropriate error recovery machinery. In the interests of simplicity we shall not give a general treatment of errors; however we shall indicate in a later section how the descriptions of the operations at the user interface can be completed.

We shall define two further operations on this structure. The first involves deleting a resource and all use of that resource. This is an operation to be treated with caution: see Theorem 7 below.

R-U-Del-Res

$$\Delta R\text{-}U$$
$$r? : R$$

$$\forall t : t? \bullet r? \in \mathrm{dom}\ ru(t) \land ru'(t) = \{ r? \} \lhd ru(t)$$

Informally, this operation may be described as follows. Consider each element t in t? and the corresponding relation ru(t) in turn. All elements (r?, u) are to be removed from ru(t).

Theorem 4

$$R\text{-}U \land [\, t? : \mathbb{P}\, T;\ r? : R \mid \forall t : t? \bullet r? \in \mathrm{dom}\ ru(t) \,] \vdash \exists R\text{-}U' \bullet R\text{-}U\text{-}Del\text{-}Res$$

Corresponding to deleting a resource there is an operation which, given a user value u?, deletes all pairs (r, u?) from the relations associated with time values in t?. This is defined as follows:

R-U-Del-User

$$\Delta R\text{-}U$$
$$u? : U$$

$$\forall t : t? \bullet u? \in \mathrm{rng}\ ru(t) \land ru'(t) = ru(t) \rhd \{ u? \}$$

Theorem 5

R-U ∧ [t? : ℙ T; u? : U | ∀ t : t? • u? ∈ rng ru(t)]
⊢
∃ R-U' • R-U-Del-User

So far we have listed theorems that a specifier is obliged to prove; viz the result that the initial state satisfies the required definition (and therefore that the specification is consistent) and the theorems that explicitly give the pre-conditions for each operation.

For the specifications that we shall develop from now on these theorems have been omitted in the interests of brevity.

In addition to these obligatory results, there are other "optional" theorems that are a consequence of the specification, and which often give insight into the structure being developed.

Two such results for our system are as follows:

Theorem 6

R-U-Book ⨾ R-U-Cancel ⊢ ru' = ru.

Informally, this theorem states that if we make a booking and follow it immediately by a cancellation using the same input parameters, then the state of the system does not change.

Theorem 7

R-U-Del-Res
⊢
in-use' = in-use ⊕ (λt:t? • in-use(t) - { r? }) ∧
users' = users ⊕
(λt:t? • users(t) - { u : U | ur(t)⦇{u}⦈ = {r?} })

This theorem makes precise the informal comment made earlier about the need for caution with the R-U-Del-Res operation. This theorem shows that resources are removed from the system structures, which we do expect, but furthermore the operation can also remove existing users.

There is a similar result concerning the R-U-Del-User operation.

5. Specialisation of the General R-U System

We shall now *specialise* the general R-U system into particular classes of the system. These specialisations are motivated by the observation that for some of the instances listed earlier, at any given time a resource may be related to only one user, or a user may occupy only one resource, or both.

5.1 An R-U system where resources cannot be shared

The first case we define is the class where each resource may be utilised by at most one user, but each user may occupy several resources. We denote this system by R⩥U (where "⩥" is just a character in the name) and define it formally by

$$R⩥U \;\widehat{=}\; [\; R\text{-}U \mid rng(ru) \subseteq R \mathrel{+\!\!\!\rightarrow} U \;]$$

The initial state of this system is given by the same condition as for Init-R-U; thus we have

$$Init\text{-}R⩥U \;\widehat{=}\; [\; R⩥U \mid rng(ru) = \{\; \{\} \;\} \;]$$

All operations are described in terms of

$$\Delta R⩥U \;\widehat{=}\; R⩥U \wedge R⩥U'$$

The operations on this system may be defined as special cases of the general operations for R-U. We first consider the booking operation.

$$R⩥U\text{-}Book \;\widehat{=}\; \Delta R⩥U \wedge [\; R\text{-}U\text{-}Book \mid \forall\, t : t? \bullet r? \notin dom\, ru(t?) \;]$$

The qualifying predicate is included in indicate that there is a further pre-condition for booking a resource in a R⩥U system.

We now have two parts to the pre-condition for this operation; firstly this qualifying predicate, and secondly the pre-condition arising from R-U-Book. In fact the former implies the latter, as is easily checked.

The cancellation operation is defined as follows:

$$R⩥U\text{-}Cancel \;\widehat{=}\; R\text{-}U\text{-}Cancel \wedge \Delta R⩥U$$

On considering the two deletion operations defined for R-U, we observe that R-U-Del-Res is equivalent to a cancellation in our present context, because the resource is associated with only one user. We therefore need only the operation which deletes a user.

$$R\rangle U\text{-}Del\text{-}User \;\widehat{=}\; R\text{-}U\text{-}Del\text{-}User \wedge \Delta R\rangle U$$

5.2 An R-U system where each user may occupy at most one resource

The second case we define is the class where each user may occupy at most one resource but resources may be shared amongst users. We denote this system by $R\langle U$ and define it formally by

$$R\langle U \;\widehat{=}\; [\; R\text{-}U \mid rng(ur) \subseteq U \rightarrow R \;]$$

The initial state of this system is also given by the predicate for Init-R-U. We have

$$Init\text{-}R\langle U \;\widehat{=}\; [\; R\langle U \mid rng(ru) = \{\; \{\} \;\} \;]$$

The operations are described in terms of

$$\Delta R\langle U \;\widehat{=}\; R\langle U \wedge R\langle U$$

We now define the booking operation for the system.

$$R\langle U\text{-}Book \;\widehat{=}\; \Delta R\langle U \wedge [\; R\text{-}U\text{-}Book \mid \forall\, t : t? \bullet u? \notin rng\, ru(t) \;]$$

As before, a qualifying predicate is needed and again as before the constraint given here implies the earlier pre-condition for the general R-U-Book operation.

The cancellation operation is defined as follows:

$$R\langle U\text{-}Cancel \;\widehat{=}\; R\text{-}U\text{-}Cancel \wedge \Delta R\langle U$$

On considering the two deletion operations defined for R-U, we observe that this time R-U-Del-User is equivalent to a cancellation in our present context, because a user may be associated with only one resource. We therefore need only the operation which deletes a resource.

$$R\langle U\text{-}Del\text{-}Res \;\widehat{=}\; R\text{-}U\text{-}Del\text{-}Res \wedge \Delta R\langle U$$

5.3 An R-U system where a user occupies at most one nonsharable resource

The third and last specialisation we define shares all the properties of the systems defined in the preceding two sections. It is therefore defined as the *conjunction* of the two schemas above. In this system each user may occupy at most one resource and each resource may be occupied by at most one user. Formally we have

$$R\exists U \triangleq R\rangle U \wedge R\langle U$$

The initial state of this system is clearly defined by

$$Init\text{-}R\exists U \triangleq [\, R\exists U \mid rng(ru) = \{ \{\} \} \,]$$

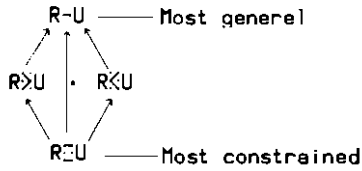
The operations on this system are given by the conjunction of the operations defined for each of the two earlier systems. For this system we require only the booking and cancellation operations. Thus we have

$$R\exists U\text{-}Book \triangleq R\rangle U\text{-}Book \wedge R\langle U\text{-}Book$$

$$R\exists U\text{-}Cancel \triangleq R\rangle U\text{-}Cancel \wedge R\langle U\text{-}Cancel$$

5.4 The specification library

We have now constructed four specifications which might be considered to form the nucleus of a specification library for resource-user systems. We may summarise the relationships between the four classes of system schematically as follows:



6. Classification and Instantiation

6.1 Some laws for CAVIAR

In this section, in order to illustrate the clarification process which took place during requirements analysis, we list some observations about the CAVIAR system which emerged during dialogue with the customer. We formalise the important constraints as *laws* which need to be taken account into account in the development which follows.

1. At any time a conference room is associated with only one meeting.

2. At any time a meeting may be associated with more than one conference room.

Law 1 is reasonably obvious: it would be difficult to hold more than one meeting in a given room. Law 2 is not obvious: it was unclear from the informal description whether or not a meeting could occupy more than one room. In fact the customer believed initially that a meeting could only take up one room, but a counter-example was found amongst the supporting documentation.

3. At any time a meeting is associated with only one dining room.

4. At any time participants from several meetings can occupy the same dining room.

These laws followed from the informal information provided that all visitors in a particular meeting would go to lunch in the same dining room. It was further established that all seats in a dining room were treated as indistinguishable, so further meetings could be accommodated if enough seats were available. Further clarification was necessary regarding lunch times: it transpired that there were "early" and "late" lunches; however this was handled by "doubling up" each dining room. For example, a booking would be made for "DR 1, early" and this was a different dining room from "DR 1, late."

5. At any time a visitor is associated with only one meeting.

6. At any time a meeting may involve several visitors.

Law 5 had to be checked out with the customer.

7. At any time a hotel room is associated with only one visitor and vice versa.

8. At any time a transport reservation is associated with only one visitor and vice versa.

Law 7 was natural, but law 8 was less so. It was established that even if the transport department decided to use a minibus, a separate transport reservation would be issued to each visitor.

6.2 Matching system with models

In this section we first consider each CAVIAR subsystem in turn and match it to the appropriate model. In fact we have enough structure available to define two subsystems directly and we do this in the remainder of this section.

(1) We first consider the conference room - meeting system CR-M.

From laws 1 and 2 we see that CR-M is an instance of the R⋗U subsystem.

(2) The dining room - meeting subsystem DR-M.

Applying laws 3 and 4 we find that DR-M is an instance of R⋖U.

However this system does not contain any information about numbers of seats or the lunch details, so we will need to extend this system later.

(3) The meeting - visito subsystem M-V.

From laws 5 and 6 M-V is an instance of R⋖U.

However we have not documented the fact that meetings have to be created before visitors can be attached to them; this will also be done later.

(4) The hotel reservation - visitor subsystem HR-V, and the transport reservation - visitor subsystem TR-V, both have the property that each resource is occupied by only one user and vice versa. Therefore both these systems are instances of R⋍U.

In fact this model is sufficient to define HR-V and TR-V completely, by *instantiation*, as we now show.

6.3 The hotel reservation subsystem - HR-V

We define HR-V as follows:

$$\text{HR-V} \;\triangleq\; \text{R}\exists\text{U}_{\text{HR-V}}[\text{Date}, \text{HR}, \text{V}]$$

This object is a *decorated* instance of the R∃U schema, with its parameter sets instantiated by the sets Date, HR and V introduced in section 2. To be more explicit, the definition above is shorthand for the following:

HR-V

$$\begin{array}{l}
ru_{\text{HR-V}} : \text{Date} \rightarrow (\text{HR} \leftrightarrow \text{V}) \\
\text{in-use}_{\text{HR-V}} : \text{Date} \rightarrow \mathbb{P}\,\text{HR} \\
\text{users}_{\text{HR-V}} : \text{Date} \rightarrow \mathbb{P}\,\text{V} \\
ur_{\text{HR-V}} : \text{Date} \rightarrow (\text{V} \leftrightarrow \text{HR}) \\
\hline
\text{rng}(ru_{\text{HR-V}}) \subseteq \text{HR} \twoheadrightarrow \text{V} \;\wedge \\
\text{rng}(ur_{\text{HR-V}}) \subseteq \text{V} \twoheadrightarrow \text{HR} \;\wedge \\
(\forall t : \text{Date};\ r : \text{HR} \bullet \\
\quad r \in \text{in-use}_{\text{HR-V}}(t) \;\Leftrightarrow\; r \in \text{dom}(ru_{\text{HR-V}}(t))\ ) \;\wedge \\
(\forall t : \text{Date};\ u : \text{V} \bullet \\
\quad u \in \text{users}_{\text{HR-V}}(t) \;\Leftrightarrow\; u \in \text{ran}(ru_{\text{HR-V}}(t))\ ) \;\wedge \\
(\forall t : \text{Date} \bullet ur_{\text{HR-V}}(t) = (ru_{\text{HR-V}}(t))^{-1})
\end{array}$$

Thus each component of the schema is given the decoration in the definition, and each occurrence of the parametrised sets is instantiated as shown above. From now on we shall use such decoration without further comment.

The initial state of HR-V is given by

$$\text{Init-HR-V} \;\triangleq\; \text{Init-R}\exists\text{U}_{\text{HR-V}}[\text{Date}, \text{HR}, \text{V}]$$

and the operations are given by

$$\text{Book-Hotel-Room}_0 \;\triangleq\; \text{R}\exists\text{U-Book}_{\text{HR-V}}[\text{Date}, \text{HR}, \text{V}]$$

and

$$\text{Cancel-Hotel-Room}_0 \;\triangleq\; \text{R}\exists\text{U-Cancel}_{\text{HR-V}}[\text{Date}, \text{HR}, \text{V}]$$

6.4 The transport reservation subsystem - TR-V

This subsystem is essentially the same as the HR-V subsystem except for the parametrisation. The instances of the parameters are denoted respectively Time, TR and V, where once again the sets TR and V are as in section 2. We shall not specify the set Time further, except to repeat that it contains a Date component (see section 2). Thus we have

$$\mathsf{TR\text{-}V} \triangleq \mathsf{R\Xi U}_{\mathsf{TR\text{-}V}}[\mathsf{Time}, \mathsf{TR}, \mathsf{V}]$$

with initial state given by

$$\mathsf{Init\text{-}TR\text{-}V} \triangleq \mathsf{Init\text{-}R\Xi U}_{\mathsf{TR\text{-}V}}[\mathsf{Time}, \mathsf{TR}, \mathsf{V}]$$

and operations given by

$$\mathsf{Book\text{-}Transport}_0 \triangleq \mathsf{R\Xi U\text{-}Book}_{\mathsf{TR\text{-}V}}[\mathsf{Time}, \mathsf{TR}, \mathsf{V}]$$

and

$$\mathsf{Cancel\text{-}Transport}_0 \triangleq \mathsf{R\Xi U\text{-}Cancel}_{\mathsf{TR\text{-}V}}[\mathsf{Time}, \mathsf{TR}, \mathsf{V}]$$

7. The Meeting Attendance Subsystem

We now turn our attention to what is necessary in order to complete a model for M-V. Booking and cancelling operations have been defined already but so far we have not taken account of the fact that before bookings can be made the system has to "create" meetings. The question of exactly which objects are "currently defined" at any particular time is important because in several cases only those objects known to the system (i. e., those objects that have been created but not yet destroyed) can book resources, etc.

7.1 A pool system

We can model this situation with a simple structure which we term a Pool. This schema is parametrised over the set T and an arbitrary set X. There are only two operations to be defined; namely those that add an object to, and delete an object from, the pool, over a specified time period.

Formally we have

$$[T, X]$$

$$\begin{array}{|l} \textit{Pool} \\ \hline exists : T \rightarrow \mathbb{P}\, X \\ \hline \end{array}$$

with initial state given by

$$Init\text{-}Pool \;\widehat{=}\; [\, Pool \mid rng(exists) = \{\, \{\} \,\} \,]$$

For later use we define

$$\equiv Pool \;\widehat{=}\; [\, \Delta Pool \mid Pool' = Pool \,]$$

Given

$$\Delta Pool \;\widehat{=}\; Pool \wedge Poll'$$

The operations are given by

Create

$$\Delta Pool$$
$$t? : \mathbb{P}\, T$$
$$x? : X$$

$$exists' = exists \oplus (\lambda\, t : t? \bullet exists(t) \cup \{ x? \})$$

and

Destroy

$$\Delta Pool$$
$$t? : \mathbb{P}\, T$$
$$x? : X$$

$$exists' = exists \oplus (\lambda\, t : t? \bullet exists(t) - \{ x? \})$$

We could have included in the Create operation the pre-condition that the object x? not already exist for any of the times in t?. However we make a deliberate decision here to omit this - having in mind the situation where an object may already exist for some of the times in t? and its existence needs to be extended to all of t?. A similar remark applies to the Destroy operation.

7.2 The meeting - visitor subsystem

To construct the model for the M-V system we combine the Pool and R⩤U structures.

M-V

$$R⩤U_{M\text{-}V}[Session, M, V]$$
$$Pool_M[Session, M]$$

$$\forall t : T \bullet$$
$$in\text{-}use_{M\text{-}V}(t) \subseteq exists_M(t)$$

Thus we have combined an M-V instance of an R⩤U system and a meeting instantiation of a Pool system (with the parameter sets as shown). The predicate assures that visitors can only attend existing meetings.

The initial state is given by

$$\mathsf{Init\text{-}M\text{-}V} \;\widehat{=}\; \mathsf{Init\text{-}R{\not\leftarrow}U}_{M\text{-}V}[\mathsf{Session}, M, V] \wedge \mathsf{Init\text{-}Pool}_{M}[\mathsf{Session}, M]$$

We now define the operations on M-V in terms of

$$\Delta\mathsf{M\text{-}V} \;\widehat{=}\; \mathsf{M\text{-}V} \wedge \mathsf{M\text{-}V}'$$

The first operation is concerned with adding a visitor to a meeting.

$$\mathsf{Add\text{-}Visitor\text{-}to\text{-}Meeting}_{0} \;\widehat{=}\; \Delta\mathsf{M\text{-}V} \wedge \equiv\mathsf{Pool}_{M}[\mathsf{Session}, M] \wedge \mathsf{R{\not\leftarrow}U\text{-}Book}_{M\text{-}V}[\mathsf{Session}, M, V]$$

When an operation is "promoted" in this way, its new pre-condition is determined as follows: the "old" pre-condition (i. e., that arising from its definition) must be conjoined with a further predicate which arises from the new invariant of the larger state. Here, for example, the pre-condition for the earlier booking operation is given in section 5.2: namely

$$\forall\, t : t?_{M\text{-}V} \bullet u?_{M\text{-}V} \notin \mathsf{rng}(ru_{M\text{-}V}(t))$$

and this must be conjoined with

$$\forall\, t : t?_{M\text{-}V} \bullet r?_{M\text{-}V} \in \mathsf{exists}_{M}(t).$$

This second predicate is a consequence of the M-V invariant.

Thus the complete pre-condition for the Add-Visitor-to-Meeting operation is given by

$$\forall\, t : t?_{M\text{-}V} \bullet u?_{M\text{-}V} \notin \mathsf{rng}(ru_{M\text{-}V}(t)) \wedge r?_{M\text{-}V} \in \mathsf{exists}_{M}(t)$$

which states that the visitor ($u?_{M\text{-}V}$) is not already attending a meeting at that time and that the meeting he is going to attend actually exists.

The second operation removes a visitor from a meeting.

$$\text{Remove-Visitor-from-Meeting}_0 \;\widehat{=}\; \Delta M\text{-}V \wedge \exists \text{Pool}_M[\text{Session}, M] \wedge R\text{-}U\text{-}\text{Cancel}_{M\text{-}V}[\text{Session}, M, V]$$

It is easy to check that the pre-condition for the Remove-Visitor-from-Meeting operation is simply the predicate which is inherited from the initial R-U-Cancel operation; namely

$$\forall\, t : t?_{M\text{-}V} \bullet (r?_{M\text{-}V}, u?_{M\text{-}V}) \in ru_{M\text{-}V}(t)$$

We now define the operations which create and cancel meetings as follows:

$$\text{Create-Meeting}_0 \;\widehat{=}\; \Delta M\text{-}V \wedge \exists R\text{-}U_{M\text{-}V}[\text{Session}, M, V] \wedge \text{Creete}_M[\text{Session}, M]$$

For the creation there is no pre-condition.

$$\begin{array}{|l}
\text{Cancel-Meeting}_0 \\
\Delta M\text{-}V \\
R\text{-}U\text{-}\text{Del-Res}_{M\text{-}V}[\text{Session}, M, V] \\
\text{Destroy}_M[\text{Session}, M] \\
\hline
t?_M = t?_{M\text{-}V} \wedge \\
x?_M = r?_{M\text{-}V} \\
\hline
\end{array}$$

The pre-conditions for cancelling a meeting arise from the original R-U-Del-Res operation, i. e., that

$$\forall\, t : t?_{M\text{-}V} \bullet r?_{M\text{-}V} \in \text{dom}(ru_{M\text{-}V}(t))$$

and secondly from the identifications required for the input parameters.

8. The Meeting Resource Subsystems

We are left with the systems CR-M and DR-M to define. We observe that both of these have further information associated with the resource-user relationship, so in order to capture this facet in our model we introduce the concept of a *diary* system.

8.1 A diary system

The diary is to record information about some elements of a set. We denote the set in question by X and the associated information by I_X. For each t, the set of elements of X for which we have information is defined as recorded(t). Once again this system is dependent on time, T.

$$[\, T,\ X,\ I_X \,]$$

Diary

$$\begin{aligned} &info : T \rightarrow (X \nrightarrow I_X) \\ &recorded : T \rightarrow \mathbb{P}\, X \\ \hline &\forall\, t : T \bullet recorded(t) = dom(info(t)) \end{aligned}$$

with initial state given by

$$Init\text{-}Diary \;\widehat{=}\; [\, Diary \mid rng(info) = \{\, \{\} \,\} \,]$$

The two operations to be defined both involve a change over a particular time period. Note that we are motivated to make this definition in order to maintain compatibility with existing systems. Formally we define

$$\Delta Diary \;\widehat{=}\; Diary \wedge Diary' \wedge [\, t? : \mathbb{P}\, T \,]$$

Add

$$\begin{aligned} &\Delta Diary \\ &x? : X \\ &i? : I_X \\ \hline &(\forall\, t : t? \bullet x? \notin recorded(t)) \wedge \\ &info' = info \oplus (\lambda\, t : t? \bullet info(t) \oplus \{\, x? \mapsto i? \,\}) \end{aligned}$$

The complementary erasure operation would remove one element (and the information associated with it) from info(t). However we note that this is a special case of the following more powerful operation.

Erase

$\Delta Diary$
$x? : T \pfun \mathbb{P}\, X$

$dom(x?) = t? \land$
$(\forall\, t : t? \bullet x?(t) \subseteq recorded(t?)) \land$
$info' = info \oplus (\lambda\, t : t? \bullet x?(t) \lhd info(t))$

8.2 The conference room booking subsystem

We are now in a position to fully specify the subsystem CR-M, by instantiation as follows:

CR-M

$R\rangle U_{CR\text{-}M}[Session, CR, M]$
$Diary_{CR}[Session, CR, SI]$

$in\text{-}use_{CR\text{-}M} = recorded_{CR}$

with initial state given by

$$Init\text{-}CR\text{-}M \;\widehat{=}\; Init\text{-}R\rangle U_{CR\text{-}M}[Session, CR, M] \land Init\text{-}Diary_{CR}[Session, CR, SI]$$

It would be more correct to regard the session information SI as being related to a meeting rather than a conference room. The reason for associating SI with conference rooms is that it contains information which is issued to the department supplying equipment for meetings, and they are concerned with the venue rather than what is to take place there.

The operations that we require for CR-M are given below. Information is recorded about each resource when it is booked, and must be erased when a cancellation takes place. The definitions use

$$\Delta CR\text{-}M \triangleq CR\text{-}M \wedge CR\text{-}M'$$

Book-Conf-Room$_O$

$$\Delta CR\text{-}M$$
$$R \gg U\text{-}Book_{CR\text{-}M}[Session, CR, M]$$
$$Add_{CR}[Session, CR, SI]$$

$$t?_{CR\text{-}M} = t?_{CR} \wedge$$
$$r?_{CR\text{-}M} = x?_{CR}$$

Cancel-Conf-Rooms$_O$

$$\Delta CR\text{-}M$$
$$R \gg U\text{-}Del\text{-}User_{CR\text{-}M}[Session, CR, M]$$
$$Erase_{CR}[Session, CR, SI]$$

$$t?_{CR\text{-}M} = t?_{CR} \wedge$$
$$(\forall t: t?_{CR\text{-}M} \bullet x?_{CR}(t) = ur_{CR\text{-}M}(t)(\{u?_{CR\text{-}M}\}))$$

The cancellation operation here deletes all conference rooms associated with a particular meeting over the specified time period. This is the operation which is most compatible with the Cancel-Meeting operation defined for M-V. However, if required, we could also define the operation that cancels just one conference room - meeting pairing.

8.3 The dining room booking subsystem

The final subsystem that we need to consider is DR-M.

The analysis so far does not take account of the fact that dining rooms have a finite capacity, so we need to extend out model. We suppose that we have been given a function

$$max\text{-}no : DR \rightarrow \mathbb{N}$$

which records this capacity and we record the number of seats in each dining room which have been reserved already.

The DR-M system is defined formally as follows:

DR-M

$$R\!\!\lhd\!\!U_{DR\text{-}M}[Date, DR, M]$$
$$Diary_{DR}[Date, M, LI]$$
$$rsvd : T \rightarrow (DR \twoheadrightarrow \mathbb{N})$$

$$users_{DR\text{-}M} = recorded_{DR} \wedge$$
$$(\forall t : Date \bullet dom(rsvd(t)) = in\text{-}use_{DR\text{-}M}(t) \wedge$$
$$(\forall r : in\text{-}use_{DR\text{-}M}(t) \bullet rsvd(t)(r) \leq max\text{-}no(r))$$
$$)$$

Observe that in this case information is associated with each *user*, and therefore the diary system takes M as its main parameter. Dining rooms that are in use have a number of seats reserved, and this number has to be within the dining room's capacity.

The initial state of DR-M is given by

$$Init\text{-}DR\text{-}M \;\widehat{=}\; Init\text{-}R\!\!\lhd\!\!U_{DR\text{-}M}[Date, DR, M] \wedge Init\text{-}Diary_{DR}[Date, M, LI]$$

The two operations that we require for this structure are *booking* a (number of seats in a) dining room and *cancelling* a lunch booking for a particular meeting. In normal circumstances, a resource (dining room) will not be subject to being taken out of service (although this occurrence is clearly easy to model if required).

Both these operations leave rsvd unchanged for time values outside the period in question; we make this part of the operation invariant.

ΔDR-M

$\Delta R \gg U_{DR\text{-}M}[Date, DR, M]$
$\Delta Diery_{DR}[Date, M, LI]$
amount? : T ⇸ ℕ

$t?_{DR\text{-}M} = t?_{DR}$ ∧
dom(amount?) = $t?_{DR\text{-}M}$ ∧
$t?_{DR\text{-}M} \lhd rsvd' = t?_{DR\text{-}M} \lhd rsvd$

Book-Dining-Room$_0$

ΔDR-M
$R \gg U\text{-}Book_{DR\text{-}M}[Date, DR, M]$
$Add_{DR}[Date, M, LI]$

$x?_{DR} = u?_{DR\text{-}M}$ ∧
(∀t : $t?_{DR\text{-}M}$ •
  rsvd(t)($r?_{DR\text{-}M}$) + amount?(t) ≤ max-no($r?_{DR\text{-}M}$) ∧
  rsvd'(t) = rsvd(t)
    ⊕ { $r?_{DR\text{-}M}$ ↦ rsvd(t)($r?_{DR\text{-}M}$) + amount?(t) }
)

Cancel-Dining-Room$_0$

ΔDR-M
$R \gg U\text{-}Cancel_{DR\text{-}M}[Date, DR, M]$
$Erase_{DR}[Date, M, LI]$

(∀t : $t?_{DR\text{-}M}$ •
  $x?_{DR}$(t) = { $u?_{DR\text{-}M}$ } ∧
  rsvd'(t) = rsvd(t)
    ⊕ { $r?_{DR\text{-}M}$ ↦ rsvd(t)($r?_{DR\text{-}M}$) - amount?(t) }
)

8.4 The visitor pool - V-P

From the informal requirements we find that visitors must be "legitimate" before they are allowed to attend meetings or have resources booked on their behalf. This requirement is easily met by introducing a visitor Pool structure, with actual parameters Date and V. Thus we define V-P as

$$V\text{-}P \;\widehat{=}\; Pool_V[Date, V]$$

with initial state given by

$$Init\text{-}V\text{-}P \;\widehat{=}\; Init\text{-}Pool_V[Date, V]$$

The operations that we require on this structure are simply those of creation and destruction of visitors. Formally we have

$$Create\text{-}Visitor_0 \;\widehat{=}\; Create_V[Date, V]$$

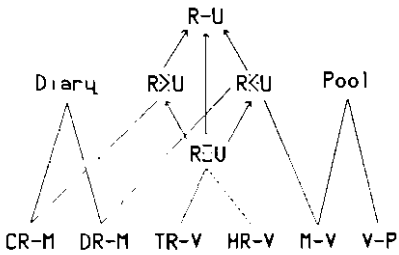
and

$$Destroy\text{-}Visitor_0 \;\widehat{=}\; Destroy_V[Date, V]$$

8.5 The construction process

In this section we summarise the constructions we have used to build the individual CAVIAR components.

In sections 7 and 8 we added *pool* and *diary* components to our basic library in section 5.4. We now have a library which consists of the 6 components R-U, R⩾U, R⩽U, R□U, Pool and Diary. We indicate in the following diagram how each subsystem has been constructed using components from the library.



9. The Complete CAVIAR System

We have now achieved our first goal of specifying all constituent subsystems of CAVIAR. We have yet to combine the subsystems into a coherent whole. This is now a comparatively easy task, once we have observed a few extra constraints.

9.1 Combining subsystems to form the system state

We define the visitor part of the system as follows:

```
V-SYS
V-P
HR-V
TR-V
-----------------------------------------------------
(∀d : Date • users_HR-V(d) ⊆ exists_V(d) ) ∧
(∀t : Time • users_TR-V(t) ⊆ exists_V(date-of-time(t)) )
```

The invariant states that visitors that have hotel or transport reservations must be known.

The meeting part of the system is defined by

```
M-SYS
M-V
CR-M
DR-M
-----------------------------------------------------
(∀s:Session • users_CR-M(s) ⊆ exists_M(s) ) ∧
(∀d:Date •
    users_DR-M(d) ⊆
    U { s:Session | date-of-session(s) = d • exists_M(s) }
)
```

The invariant states that meetings which are occupying conference rooms or dining rooms must be known to the system at that time.

These two subsystems are now combined to form the CAVIAR system.

CAVIAR

V-SYS

M-SYS

$\forall s : Session \bullet users_{M\text{-}V}(s) \subseteq exists_{V}(date\text{-}of\text{-}session(s))$

Informally, the invariant states that all visitors who are attending meetings must be known to the system.

The initial state of the system is given by the conjunction of all the initialisations. It is easy to verify that this conjunction satisfies the invariant.

$$Init\text{-}CAVIAR \triangleq Init\text{-}HR\text{-}V \wedge Init\text{-}TR\text{-}V \wedge Init\text{-}M\text{-}V \wedge Init\text{-}CR\text{-}M \wedge Init\text{-}DR\text{-}M \wedge Init\text{-}V\text{-}P$$

9.2 Operations on CAVIAR

The operations on CAVIAR may be divided naturally into three groups.

9.2.1 Operations which involve meetings only

These operations are concerned with M-SYS only and leave V-SYS unchanged. We denote this by

$$M\text{-}OP \triangleq \Delta CAVIAR \wedge \equiv V\text{-}SYS$$

where

$$\Delta CAVIAR \triangleq CAVIAR \wedge CAVIAR'$$

and

$$\equiv V\text{-}SYS \triangleq [\, V\text{-}SYS \wedge V\text{-}SYS' \mid V\text{-}SYS = V\text{-}SYS' \,]$$

(Note: in the following similar definitions of ≡CR-M, ≡DR-M, etc. are omitted.)

The first operation is to construct a meeting

$$\mathtt{Create\text{-}Meeting} \;\widehat{=}\; \mathtt{M\text{-}OP} \wedge \mathtt{Create\text{-}Meeting}_0 \wedge \Xi\mathtt{CR\text{-}M} \wedge \Xi\mathtt{DR\text{-}M}$$

This operation has no pre-condition (there is no pre-condition for $\mathtt{Create\text{-}Meeting}_0$), so it is total. The next operation is to cancel a meeting.

$$\mathtt{Cancel\text{-}Meeting}_1 \;\widehat{=}\; \mathtt{M\text{-}OP} \wedge \mathtt{Cancel\text{-}Meeting}_0 \wedge \Xi\mathtt{CR\text{-}M} \wedge \Xi\mathtt{DR\text{-}M}$$

We can determine the pre-condition for this operation as follows: first we establish the constraint arising from the system invariant. The operation removes an element from $\mathtt{exists}_M$ so this element cannot be a user in CR-M or DR-M during the period $t?_M$. Formally, we require that

$$\forall\, t : t?_M \bullet r?_{M\text{-}U} \notin \mathtt{users}_{CR\text{-}M}(t) \cup \mathtt{users}_{DR\text{-}M}(\mathtt{date\text{-}of\text{-}session}(t))$$

The second part of the pre-condition arises from the earlier pre-condition for $\mathtt{Cancel\text{-}Meeting}_0$. This is precisely

$$t?_M = t?_{M\text{-}U} \;\wedge\; x?_M = r?_{M\text{-}U} \wedge (\forall t : t?_{M\text{-}U} \bullet r?_{M\text{-}U} \in \mathtt{dom}(ru_{M\text{-}U}(t)))\,.$$

We shall at this point fulfil the promise made in section 4.1: indicating how to define the corresponding total operation. This is formed by the *disjunct* of the successful operation with the schema which takes as its qualifying predicate the *negation* of the pre-condition established above.

Cancel-Meeting-Fail

ΞCAVIAR
$t?_{M\text{-}U} : \mathbb{P}\ \mathtt{Session}$
$t?_M : \mathbb{P}\ \mathtt{Session}$
$x?_M : M$
$r?_{M\text{-}U} : M$

$$\begin{aligned}
&(\exists\, t : t?_M \bullet \\
&\quad r?_{M\text{-}U} \in \mathtt{users}_{CR\text{-}M}(t) \cup \mathtt{users}_{DR\text{-}M}(\mathtt{date\text{-}of\text{-}session}(t)))\\
&\vee\ t?_M \neq t?_{M\text{-}U}\\
&\vee\ x?_M \neq r?_{M\text{-}U}\\
&\vee\ (\exists t : t?_{M\text{-}U} \bullet r?_{M\text{-}U} \notin \mathtt{dom}(ru_{M\text{-}U}(t)))
\end{aligned}$$

$$\text{Cancel-Meeting} \triangleq \text{Cancel-Meeting}_1 \vee \text{Cancel-Meeting-Fail}$$

Informally, if the required pre-condition for the meeting cancellation is not satisfied, the system is unchanged. In practice we would require an appropriate error message to be output.

For the sake of brevity, we shall present the remainder of the operations without going through this process.

The next two operations add visitors to, and delete visitors from, a meeting.

$$\text{Add-Visitor-to-Meeting} \triangleq \text{M-OP} \wedge \text{Add-Visitor-to-Meeting}_0 \wedge \equiv\text{CR-M} \wedge \equiv\text{DR-M}$$

$$\text{Remove-Visitor-from-Meeting} \triangleq \text{M-OP} \wedge \text{Remove-Visitor-from-Meeting}_0 \wedge \equiv\text{CR-M} \wedge \equiv\text{DR-M}$$

The pre-conditions for these operations are straightforward to determine in the usual way and we shall omit them and also those for the remaining operations.

The next two operations deal with conference rooms.

$$\text{Book-Conf-Room} \triangleq \text{M-OP} \wedge \equiv\text{M-V} \wedge \text{Book-Conf-Room}_0 \wedge \equiv\text{DR-M}$$

$$\text{Cancel-Conf-Room} \triangleq \text{M-OP} \wedge \equiv\text{M-V} \wedge \text{Cancel-Conf-Room}_0 \wedge \equiv\text{DR-M}$$

We now have the two operations concerning dining rooms.

$$\text{Book-Dining-Room} \triangleq \text{M-OP} \wedge \equiv\text{M-V} \wedge \equiv\text{CR-M} \wedge \text{Book-Dining-Room}_0$$

$$\text{Cancel-Dining-Room} \triangleq \text{M-op} \wedge \equiv\text{M-V} \wedge \equiv\text{CR-M} \wedge \text{Cancel-Dining-Room}_0$$

There is one final operation to be defined in this section: namely the cancellation of both dining room and conference room(s) associated with a particular meeting. This is not the conjunct of the two cancellation operations already given because each of these leaves the components it is not acting on fixed. Hence we need a different operation defined by

$$\text{Cancel-Meeting-Arrangements} \triangleq \text{M-OP} \wedge \equiv\text{M-V} \wedge \text{Cancel-Conf-Room}_0 \wedge \text{Cancel-Dining-Room}_0$$

9.2.2 Operations which involve visitors only

This section contains operations which involve V-SYS only and leave M-SYS unchanged. We denote this group by

$$\text{V-OP} \;\widehat{=}\; \Delta\text{CAVIAR} \;\wedge\; \equiv\text{M-SYS}$$

The first pair of operations introduce visitors to and remove visitors from the visitor system.

$$\text{Create-Visitor} \;\widehat{=}\; \text{V-OP} \wedge \text{Create-Visitor}_0 \;\wedge\; \equiv\text{HR-V} \;\wedge\; \equiv\text{TR-V}$$

$$\text{Destroy-Visitor} \;\widehat{=}\; \text{V-OP} \wedge \text{Destroy-Visitor}_0 \wedge \equiv\text{HR-V} \;\wedge\; \equiv\text{TR-V}$$

The Caviar invariant induces the following pre-condition for the Destroy operation.

$$\begin{aligned}\forall t : t?_V \bullet x?_V \notin\ & \text{users}_{\text{HR-V}}(t) \\ & \cup \bigcup \{\, t : \text{date-of-time}^{-1}(\!|t?_V|\!) \bullet \text{users}_{\text{TR-V}}(t) \,\} \\ & \cup \bigcup \{\, s : \text{date-of-session}^{-1}(\!|t?_V|\!) \bullet \text{users}_{\text{M-V}}(s) \,\}\end{aligned}$$

The two operations concerned with hotel rooms are as follows:

$$\text{Book-Hotel-Room} \;\widehat{=}\; \text{V-OP} \wedge \equiv\text{V-P} \wedge \text{Book-Hotel-Room}_0 \;\wedge\; \equiv\text{TR-V}$$

$$\text{Cancel-Hotel-Room} \;\widehat{=}\; \text{V-OP} \wedge \equiv\text{V-P} \wedge \text{Cancel-Hotel-Room}_0 \wedge \equiv\text{TR-V}$$

The two operations concerned with transport reservations are

$$\text{Book-Transport} \;\widehat{=}\; \text{V-OP} \wedge \equiv\text{V-P} \wedge \equiv\text{HR-V} \wedge \text{Book-Transport}_0$$

$$\text{Cancel-Transport} \;\widehat{=}\; \text{V-OP} \wedge \equiv\text{V-P} \wedge \equiv\text{HR-V} \wedge \text{Cancel-Transport}_0$$

9.2.3 A general visitor removal operation

Finally we define an operation which removes a visitor entirely from the system for a particular set of dates.

Delete-Visitor

ΔCAVIAR
$\equiv$CR-M
$\equiv$DR-M
Cancel-Hotel-Room$_0$
Cancel-Transport$_0$
Remove-Visitor-from-Meeting$_0$
Destroy-Visitor$_0$

$x?_V = u?_{HR\text{-}V} = u?_{TR\text{-}V} = u?_{M\text{-}V}$ $\wedge$
$t?_V = t?_{HR\text{-}V}$ $\wedge$
$t?_{TR\text{-}V} = \{ d : t?_V; t : Time \mid date\text{-}of\text{-}time(t) = d$
$\wedge\ u?_{TR\text{-}V} \in users_{TR\text{-}V}(t) \bullet t \}$ $\wedge$
$t?_{M\text{-}V} = \{ d : t?_V; s : Session \mid date\text{-}of\text{-}session(s) = d$
$\wedge\ u?_{M\text{-}V} \in users_{M\text{-}V}(s) \bullet s \}$

10. Conclusion

This specification has created a conceptual model for the CAVIAR system which provides a precise description of the system state and its external interface, together with an exact functional specification of every operation. The subtle inter-relationships between constituent subsystems are described in the predicates which constrain the combination of these subsystems, and these have been taken into account in the specification of the operations. The system designer can now concentrate on the important parts of the design task: namely selecting appropriate data structures and algorithms, without having to be simultaneously concerned with the complexity of subsystem interactions. This reflects the classical principle of *separation of concerns*.

It may be argued that a specification such as we have given above is a long way from an actual software product. Experience shows however that minimal effort is required to develop software once such a specification has been constructed. For example, in the case of CAVIAR, a Pascal implementation was constructed directly and quickly from the specification.

11. Acknowledgements

A formal specification of CAVIAR was given in 1981 by J.-R. Abrial. This work was carried out at the Programming Research Group at Oxford University in collaboration with B. Sufrin, T. Clement and one of the co-authors. T. Clement implemented a prototype version of the specification on a ITT-2020 computer in UCSD Pascal. J.-R. Abrial's original specification document listed most of the properties of the system that appear in this document, though the style of the presentation, the notation, and the conventions used in this paper have since been developed by members of the Programming Research Group.

We would like to thank J.-R. Abrial for his original contribution, I. Hayes for editing this paper and all those involved in helping with the project, particularly the personnel in the Visitor Services Department of STL, who willingly provided the team with information about the current manual system in operation at that time.

We would also like to thank Bernie Cohen, Tim Denvir and Tom Cox for their initial effort in setting up this collaborative effort between STL and the Programming Research Group and their continuing interest.

12. References and Related Work

1. Abrial, J.-R. The specification language Z: Basic library. *Oxford University Programming Research Group internal report*, (April 1980).

2. Morgan, C. C. Schemas in Z: A preliminary reference manual. *Oxford University Programming Research Group Distributed Computing Project report*, (March 1984).

3. Sufrin, B. A., Sørensen, I. H., Morgan, C. C., and Hayes, I. J. Notes for a Z Handbook. *Oxford University Programming Research Group internal report*, (July 1985).

4. Morgan, C. C., and Sufrin, B. A. Specification of the UNIX file system. *IEEE Transactions on Software Engineering, Vol. 10, No. 2*, (March 1984), pp. 128-142.

5. Hayes, I. J. Specification Case Studies. *Oxford University Programming Research Group Monograph, PRG-46*, (July 1985).

Z Reference Card
Mathematical Notation
Version 2.2

Programming Research Group
Oxford University

1. Definitions and declarations.
Let x, x_k be identifiers and T, T_k sets.

$LHS \triangleq RHS$ Definition of LHS as syntactically equivalent to RHS.
$x : T$ Declaration of x as type T.
$x_1 : T_1 ; x_2 : T_2 ; \ldots ; x_n : T_n$ List of declarations.
$x_1, x_2, \ldots, x_n : T$ $\triangleq x_1 : T ; x_2 : T ; \ldots ; x_n : T$.
$[A, B]$ Introduction of generic sets.

2. Logic.
Let P, Q be predicates and D declarations.

$\mathtt{true}, \mathtt{false}$ Logical constants.
$\neg P$ Negation: "not P".
$P \wedge Q$ Conjunction: "P and Q".
$P \vee Q$ Disjunction: "P or Q".
$P \Rightarrow Q$ Implication: "P implies Q" or "if P then Q".
$P \Leftrightarrow Q$ Equivalence: "P is logically equivalent to Q".
$\forall x : T \bullet P$ Universal quantification: "for all x of type T, P holds".
$\exists x : T \bullet P$ Existential quantification: "there exists an x of type T such that P".
$\exists! x : T \bullet P_x$ Unique existence: "there exists a unique x of type T such that P". $\triangleq (\exists x : T \bullet P_x \wedge \neg(\exists y : T \mid y \neq x \bullet P_y))$
$\forall x_1 : T_1 ; x_2 : T_2 ; \ldots ; x_n : T_n \bullet P$ "For all x_1 of type T_1, x_2 of type T_2, ..., and x_n of type T_n, P holds.
$\exists x_1 : T_1 ; x_2 : T_2 ; \ldots ; x_n : T_n \bullet P$ Similar to $\forall$.
$\exists! x_1 : T_1 ; x_2 : T_2 ; \ldots ; x_n : T_n \bullet P$ Similar to $\forall$.
$\forall D \mid P \bullet Q \triangleq (\forall D \bullet P \Rightarrow Q)$.
$\exists D \mid P \bullet Q \triangleq (\exists D \bullet P \wedge Q)$.
$t_1 = t_2$ Equality between terms.
$t_1 \neq t_2 \triangleq \neg(t_1 = t_2)$.

3. Sets.
Let S, T and X he sets; t, t_k terms; P a predicate and D declarations.

$t \in S$ Set membership: "t is an element of S".
$t \notin S$ $\triangleq \neg(t \in S)$.
$S \subseteq T$ Set inclusion: $\triangleq (\forall x : S \bullet x \in T)$.
$S \subset T$ Strict set inclusion: $\triangleq S \subseteq T \wedge S \neq T$.
$\{\}$ The empty set.
$\{ t_1, t_2, \ldots, t_n \}$ The set containing $t_1, t_2, \ldots$ and t_n.
$\{ x : T \mid P \}$ The set containing exactly those x of type T for which P holds.
$(t_1, t_2, \ldots, t_n)$ Ordered n-tuple of $t_1, t_2, \ldots$ and t_n.
$T_1 \times T_2 \times \ldots \times T_n$ Cartesian product: the set of all n-tuples such that the kth component is of type T_k.
$\{ x_1 : T_1 ; x_2 : T_2 ; \ldots ; x_n : T_n \mid P \}$ The set of n-tuples $(x_1, x_2, \ldots, x_n)$ with each x_k of type T_k such that P holds.
$\{D \mid P \bullet t\}$ The set of t's such that given the declarations D, P holds.

$\{D \cdot t\}$	$\triangleq \{ D \mid true \cdot t \}$.
$\mathbb{P}\, S$	Powerset: the set of all subsets of S.
$\mathbb{F}\, S$	Set of finite subsets of S: $\triangleq \{ T: \mathbb{P}\, S \mid T \text{ is finite} \}$.
$S \cap T$	Set intersection: given S, T: $\mathbb{P}\, X$, $\triangleq \{ x:X \mid x \in S \wedge x \in T \}$.
$S \cup T$	Set union: given S, T: $\mathbb{P}\, X$, $\triangleq \{ x:X \mid x \in S \vee x \in T \}$.
$S - T$	Set difference: given S, T: $\mathbb{P}\, X$, $\triangleq \{ x:X \mid x \in S \wedge x \notin T \}$.
$\cap\, SS$	Distributed set intersection: given SS: $\mathbb{P}\, (\mathbb{P}\, X)$, $\triangleq \{x:X \mid (\forall S:SS \cdot x \in S)\}$.
$\cup\, SS$	Distributed set union: given SS: $\mathbb{P}\, (\mathbb{P}\, X)$, $\triangleq \{x:X \mid (\exists S:SS \cdot x \in S)\}$.
$\lvert S \rvert$	Size (number of distinct elements) of a finite set.
$\#S$	$\triangleq \lvert S \rvert$.

4. Numbers.

$\mathbb{N}$	The set of natural numbers (non-negative integers).
$\mathbb{N}^+$	The set of strictly positive natural numbers: $\triangleq \mathbb{N} - \{ 0 \}$.
$\mathbb{Z}$	The set of integers (positive, zero and negative).
$m..n$	The set of integers between m and n inclusive: $\triangleq \{ k:\mathbb{Z} \mid m \leq k \wedge k \leq n \}$.
$min\ S$	Minimum of a set, $S : \mathbb{F}\, \mathbb{N}$. $min\ S \in S \wedge (\forall x : S \cdot x \geq min\ S)$.
$max\ S$	Maximum of a set, $S : \mathbb{F}\, \mathbb{N}$. $max\ S \in S \wedge (\forall x : S \cdot x \leq max\ S)$.

5. Relations.

A relation is modelled by a set of ordered pairs hence operators defined for sets can be used on relations.

Let X, Y, and Z be sets; $x : X$; $y : Y$; and $R : X \leftrightarrow Y$.

$X \leftrightarrow Y$	The set of relations from X to Y: $\triangleq \mathbb{P}\, (X \times Y)$.
$x\ R\ y$	x is related by R to y: $\triangleq (x, y) \in R$.
$x \mapsto y$	$\triangleq (x, y)$
$\{ x_1 \mapsto y_1, x_2 \mapsto y_2, \ldots, x_n \mapsto y_n \}$	The relation $\{ (x_1, y_1), \ldots, (x_n, y_n) \}$ relating x_1 to y_1, . . . , and x_n to y_n.
$dom\ R$	The domain of a relation: $\triangleq \{x:X \mid (\exists y:Y \cdot x\ R\ y)\}$.
$rng\ R$	The range of a relation: $\triangleq \{y:Y \mid (\exists x:X \cdot x\ R\ y)\}$.
$R_1 \fatsemi R_2$	Forward relational composition: given $R_1: X \leftrightarrow Y$; $R_2: Y \leftrightarrow Z$, $\triangleq \{ x:X;\ z:Z \mid (\exists y:Y \cdot x\ R_1\ y \wedge y\ R_2\ z) \}$.
$R_1 \circ R_2$	Relational composition: $\triangleq R_2 \fatsemi R_1$.
R^{-1}	Inverse of relation R: $\triangleq \{ y:Y;\ x:X \mid x\ R\ y \}$.
$id\ X$	Identity function on the set X: $\triangleq \{ x : X \cdot x \mapsto x \}$.
R^k	The relation R composed with itself k times: given $R : X \leftrightarrow X$, $R^0 \triangleq id\ X$, $R^{k+1} \triangleq R^k \circ R$.
R^*	Reflexive transitive closure: $\triangleq \cup \{ n: \mathbb{N} \cdot R^n \}$.
R^+	Non-reflexive transitive closure: $\triangleq \cup \{ n: \mathbb{N}^+ \cdot R^n \}$.
$R(\!\lvert S \rvert\!)$	Image: given $S : \mathbb{P}\, X$, $\triangleq \{y:Y \mid (\exists x:S \cdot x\ R\ y)\}$.

$S \lhd R$ Domain restriction to S:
given $S : \mathbb{P} X$,
$\triangleq \{x:X; y:Y \mid x \in S \wedge x\ R\ y\}$.

$S \lhd\!\!\!\!- R$ Domain subtraction:
given $S : \mathbb{P} X$,
$\triangleq (X - S) \lhd R$.

$R \rhd T$ Range restriction to T:
given $T : \mathbb{P} Y$,
$\triangleq \{x:X; y:Y \mid x\ R\ y \wedge y \in T\}$.

$R \rhd\!\!\!\!- T$ Range subtraction of T:
given $T : \mathbb{P} Y$,
$\triangleq R \rhd (Y - T)$.

$R_1 \oplus R_2$ Overriding: given $R_1, R_2 : X \leftrightarrow Y$,
$\triangleq (\mathrm{dom}\ R_2 \lhd\!\!\!\!- R_1) \cup R_2$.

6. Functions.

A function is a relation with the property that for each element in its domain there is a unique element in its range related to it. As functions are relations all the operators defined above for relations also apply to functions.

$X \pfun Y$ The set of partial functions from X to Y:
$\triangleq \{ f : X \leftrightarrow Y \mid (\forall x : \mathrm{dom}\ f \bullet (\exists! y : Y \bullet x\ f\ y)) \}$.

$X \fun Y$ The set of total functions from X to Y:
$\triangleq \{ f : X \pfun Y \mid \mathrm{dom}\ f = X \}$.

$X \pinj Y$ The set of one-to-one partial functions from X to Y:
$\triangleq \{ f : X \pfun Y \mid (\forall y : \mathrm{rng}\ f \bullet (\exists! x : X \bullet x\ f\ y)) \}$.

$X \rightarrowtail Y$ The set of one-to-one total functions from X to Y:
$\triangleq \{ f : X \pinj Y \mid \mathrm{dom}\ f = X \}$.

$f\ t$ The function f applied to t.

$(\lambda\ x : X \mid P \bullet t)$
Lambda-abstraction:
the function that given an argument x of type X such that P holds the result is t.
$\triangleq \{ x : X \mid P \bullet x \mapsto t \}$.

$(\lambda\ x_1 : T_1; \ldots; x_n : T_n \mid P \bullet t)$
$\triangleq \{x_1 : T_1; \ldots; x_n : T_n \mid P \bullet (x_1, \ldots, x_n) \mapsto t \}$.

7. Orders.

`partial_order X`
The set of partial orders on X.
$\triangleq \{ R : X \leftrightarrow X \mid \forall x, y, z : X \bullet$
$x\ R\ x \wedge$
$x\ R\ y \wedge y\ R\ x \Rightarrow x = y \wedge$
$x\ R\ y \wedge y\ R\ z \Rightarrow x\ R\ z \}$.

`total_order X`
The set of total orders on X.
$\triangleq \{ R : \mathrm{partial\_order}\ X \mid \forall x, y : X \bullet x\ R\ y \vee y\ R\ x \}$.

`monotonic` $X\ <_X$
The set of functions from X to X that are monotonic with respect to the order $<_X$ on X.
$\triangleq \{ f : X \pfun X \mid x <_X y \Rightarrow f(x) <_X f(y) \}$.

8. Sequences.

`seq X` The set of sequences whose elements are drawn from X:
$\triangleq$ { A: $\mathbb{N}^+ \rightarrow\!\!\!\!\!\!+\;\; X$ |
dom A = 1..|A| }.

`|A|` The length of sequence A.

`[]` The empty sequence {}.

$[a_1, \ldots, a_n]$
$\triangleq$ { $1 \mapsto a_1, \ldots, n \mapsto a_n$ }.

$[a_1, \ldots, a_n] \frown [b_1, \ldots, b_m]$
Concatenation:
$\triangleq$ $[a_1, \ldots, a_n, b_1, \ldots, b_m]$,
$[] \frown A = A \frown [] = A$.

`head A` $\triangleq$ A(1).

`last A` $\triangleq$ A(|A|).

`tail` $[x] \frown A \triangleq A$.

`front` $A \frown [x] \triangleq A$.

`rev` $[a_1, a_2, \ldots, a_n]$
Reverse:
$\triangleq$ $[a_n, \ldots, a_2, a_1]$,
rev [] = [].

$\frown/AA$ Distributed concatenation:
given AA : seq(seq(X)),
$\triangleq$ $AA(1) \frown \ldots \frown AA(|AA|)$,
$\frown/[] = []$.

⨾/AR Distributed relational composition:
given AR : seq (X $\leftrightarrow$ X),
$\triangleq$ AR(1) ⨾ ... ⨾ AR(|AR|),
⨾/[] = id X.

`disjoint` AS Pairwise disjoint:
given AS: seq (ℙ X),
$\triangleq$ ($\forall$ i, j : dom AS • i $\neq$ j
$\Rightarrow$ AS(i) $\cap$ AS(j) = {}).

AS `partitions` S
$\triangleq$ disjoint AS
$\wedge$ $\cup$ ran AS = S.

A `in` B Contiguous subsequence:
$\triangleq$ ($\exists$C, D: seq X •
$C \frown A \frown D = B$).

`squash f` Convert a function, f: $\mathbb{N} \rightarrow\!\!\!\!\!\!+\;\; X$, into a sequence by squashing its domain.
squash {} = [],
and if $f \neq \{\}$ then
squash f =
$[f(i)] \frown$ squash($\{i\} \lhd f$)
where i = min(dom f) e.g.
squash {$2 \mapsto$A, $27 \mapsto$C, $4 \mapsto$B}
= [A, B, C]

S ⩤ A Restrict the sequence A to those items whose index is in the set S:
$\triangleq$ squash(S $\lhd$ A)

A ⩥ T Restrict the range of the sequence A to the set T:
$\triangleq$ squash(A $\rhd$ T).

9. Bags.

`bag X` The set of bags whose elements are drawn from X:
$\triangleq$ $X \rightarrow\!\!\!\!+\;\; \mathbb{N}^+$
A bag is represented by a function that maps each element in the bag onto its frequency of occurrence in the bag.

`[]` The empty bag {}.

$[\![x_1, x_2, \ldots, x_n]\!]$ The bag containing x_1, x_2, ... and x_n with the frequency they occur in the list.

`items s` The bag of items contained in the sequence s:
$\triangleq$ { x:rng s •
$x \mapsto |\{i : \text{dom } s \mid s(i) = x\}|$
}

Z Reference Card
Schema Notation
[For details see "Schemas in Z"]

Programming Research Group
Oxford University

Schema definition: a schema groups together some declarations of variables and a predicate relating these variables. There are two ways of writing schemas: vertically, for example

S

$$x : \mathbb{N}$$
$$y : \mathrm{seq}\ \mathbb{N}$$
$$x \leq |y|$$

or horizontally, for the same example

$S \mathrel{\widehat=} [\, x: \mathbb{N};\ y: \mathrm{seq}\ \mathbb{N} \mid x \leq |y| \,]$.

Use in signatures after $\forall$, λ, $\{\ldots\}$, etc.:

$(\forall S \bullet y \neq [\,]) \mathrel{\widehat=} (\forall x:\mathbb{N};\ y: \mathrm{seq}\ \mathbb{N} \mid x \leq |y| \bullet y \neq [\,])$.

`tuple S` The tuple formed of a schema's variables.

`pred S` The predicate part of a schema: e.g. `pred S` is $x \leq |y|$.

Inclusion A schema S may be included within the declarations of a schema T, in which case the declarations of S are merged with the other declarations of T (variables declared in both S and T must be the same type) and the predicates of S and T are conjoined. e.g.

T

$$S$$
$$z : \mathbb{N}$$
$$z < x$$

is

$$x, z : \mathbb{N}$$
$$y : \mathrm{seq}\ \mathbb{N}$$
$$x \leq |y| \land z < x$$

`S | P` The schema S with P conjoined to its predicate part. e.g. `(S | x>0)` is $[x:\mathbb{N}; y:\mathrm{seq}\ \mathbb{N} \mid x \leq |y| \land x>0]$.

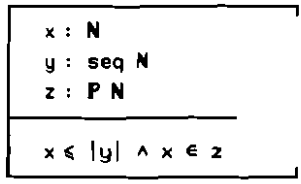
`S ; D` The schema S with the declarations D merged with the declarations of S. e.g. `(S ; z : N)` is $[\, x,z:\mathbb{N};\ y:\mathrm{seq}\ \mathbb{N} \mid x \leq |y| \,]$

`S[new/old]` Renaming of components: the schema S with the component `old` renamed to `new` in its declaration and every free use of that `old` within the predicate. e.g. `S[z/x]` is $[\, z:\mathbb{N};\ y:\mathrm{seq}\ \mathbb{N} \mid z \leq |y| \,]$ and `S[y/x, x/y]` is $[\, y:\mathbb{N};\ x:\mathrm{seq}\ \mathbb{N} \mid y \leq |x| \,]$

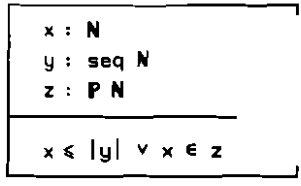
Decoration Decoration with subscript, superscript, prime, etc.; systematic renaming of the variables declared in the schema. e.g. S′ is $[x':\mathbb{N};\ y':\mathrm{seq}\ \mathbb{N} \mid x' \leq |y'|]$

`¬S` The schema S with its predicate part negated. e.g. `¬S` is $[x:\mathbb{N};\ y:\mathrm{seq}\ \mathbb{N} \mid \neg(x \leq |y|)]$

`S ∧ T` The schema formed from schemas S and T by merging their declarations (see inclusion above) and and'ing their predicates. Given $T \mathrel{\widehat=} [x: \mathbb{N};\ z: \mathbb{P}\,\mathbb{N} \mid x \in z]$, `S ∧ T` is



$S \vee T$ The schema formed from schemas S and T by merging their declarations and or'ing their predicates. e.g. $S \vee T$ is

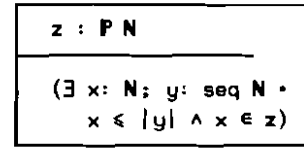


$S \Rightarrow T$ The schema formed from schemas S and T by merging their declarations and taking pred S ⇒ pred T as the predicate. e.g. $S \Rightarrow T$ is similar to $S \wedge T$ and $S \vee T$ except the predicate contains an "⇒" rather than an "∧" or an "∨".

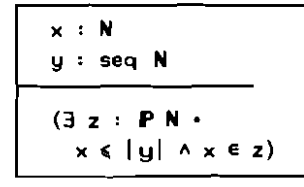
$S \Leftrightarrow T$ The schema formed from schemas S and T by merging their declarations and taking pred S ⇔ pred T as the predicate. e.g. $S \Leftrightarrow T$ the same as $S \wedge T$ with "⇔" in place of the "∧".

$S \setminus (v_1, v_2, \ldots, v_n)$
Hiding: the schema S with the variables $v_1, v_2, \ldots,$ and v_n hidden: the variables listed are removed from the declarations and are existentially quantified in the predicate. e.g. $S \setminus x$ is $[y : \text{seq } \mathbb{N} \mid (\exists x : \mathbb{N} \bullet x \leqslant |y|)]$

A schema may be specified instead of a list of variables; in this case the variables declared in that schema are hidden.
e.g. $(S \wedge T) \setminus S$ is



$S \upharpoonright (v_1, v_2, \ldots, v_n)$
Projection: The schema S with any variables that do not occur in the list $v_1, v_2, \ldots, v_n$ hidden: the variables removed from the declarations are existentially quantified in the predicate.
e.g. $(S \wedge T) \upharpoonright (x, y)$ is



The list of variables may be replaced by a schema as for hiding; the variables declared in the schema are used for the projection.

The following conventions are used for variable names in those schemas which represent operations:

undashed	state before the operation,
dashed	state after the operation,
ending in "?"	inputs to the operation, and
ending in "!"	outputs from the operation.

The following schema operations only apply to schemas following the above conventions.

pre S Precondition: all the state after components (dashed) and the outputs (ending in "!") are hidden. e.g. given

S

$$x?,\ s,\ s',\ y! : \mathbb{N}$$
$$s' = s - x? \land y! = s$$

pre S is

$$x?,\ s : \mathbb{N}$$
$$(\exists\, s',\ y! : \mathbb{N} \bullet\ s' = s - x? \land y! = s)$$

post S Postcondition: this is similar to precondition except all the state before components (undashed) and inputs (ending in "?") are hidden.

S ⊕ T Overriding:
≙ (S ∧ ¬pre T) ∨ T.
e.g. given S above and

T

$$x?,\ s,\ s' : \mathbb{N}$$
$$s < x? \land s' = s$$

S ⊕ T is

$$x?,\ s,\ s',\ y! : \mathbb{N}$$
$$(s' = s - x? \land y! = s \land \neg(\exists\, s' : \mathbb{N} \bullet\ s < x? \land s' = s)) \lor (s < x? \land s' = s)$$

The predicate can be simplified:

$$x?,\ s,\ s',\ y! : \mathbb{N}$$
$$(s' = s - x? \land y! = s \land s \geqslant x?) \lor (s < x? \land s' = s)$$

S ⨾ T Schema composition: if we consider an intermediate state that is both the final state of the operation S and the initial state of the operation T then the composition of S and T is the operation which relates the initial state of S to the final state of T through the intermediate state. To form the composition of S and T we take the state after components of S and the state before components of T that have a basename* in common, rename both to new variables, take the schema "and" (∧) of the resulting schemas, and hide the new variables.
e.g. S ⨾ T is

$$x?,\ s,\ s',\ y! : \mathbb{N}$$
$$(\exists\, s_0 : \mathbb{N} \bullet\ s_0 = s - x? \land y! = s \land s_0 < x? \land s' = s_0)$$

* basename is the name with any decoration ("'", "!", "?",etc.) removed.

S >> T	Piping: this schema operation is similar to schema composition; the difference is that rather than identifying the state after components of S with the state before components of T, the output components of S (ending in "!") are identified with the input components of T (ending in "?") that have the same basename.