Grey Box Data Refinement

Eerke Boiten and John Derrick

Computing Laboratory, University of Kent
Canterbury, CT2 7NF, U.K.
(Phone: +44 1227 764000,
Email: E.A.Boiten@ukc.ac.uk)

Abstract. We introduce the concepts of grey box and display box data types. These make explicit the idea that state variables in abstract data types are not always hidden. Programming languages have visibility rules which make representations observable and modifiable. Specifications in model-based notations may have implicit assumptions about visible state components, or are used in contexts where the representation *does* matter. Grey box data types are like the "standard" black box data types, except that they contain explicit subspaces of the state which are modifiable and observable. Display boxes indirectly observe the state by adding displays to a black box. Refinement rules for both these alternative data types are given, based on their interpretations as black boxes.

1 Introduction

Programming languages that support modularisation and encapsulation of data types with their operations have various ways of dealing with the variables that represent the "state" of the data type. The method which is most often adopted by specification language designers and other theorists is the one where all state components are invisible to any program part outside the encapsulated data type ("black box", "representation hiding"). This gives the cleanest semantics, the most explicit interface, and the fewest headaches in terms of reuse and reimplementation. Some object-oriented programming languages, e.g. Smalltalk [7], and most model-oriented specification languages, like Z [17, 19], take this approach.

However, representation hiding is conceptually nice but in practice sometimes cumbersome. For example, object-oriented languages have to deal with the problem of *binary methods* [4]: how to view and implement an operation that conceptually takes two abstract objects as its input, given that neither of the objects should be seeing the other's representation? In C++ for example, this has given rise to the notion of *friends*, with complicated visibility rules. In any case, having an explicit distinction between "private" and "public" components as in C++ or Java reduces the complexity of the specification of interfaces: no explicit functions for observing and modifying public components have to be specified. A consequence of having visible components is also that they need to be preserved in inheritance – which may be viewed as a kind of data refinement. If one wants to develop executable programs from formal specifications, it is useful if the formal specification notation has features which approximate those of the programming language. In that context, it is important to note that "observability" in specification languages is a weaker notion than "visibility" in programming languages: the latter normally implies "modifiability" as well.

Refinement rules [8] for model-oriented languages have been derived with the black box style of specification in mind. However, users of model-oriented specification languages actually do not always assume their representations to be invisible, even if they use the "states and operations" style. For example, the specification of an editor in [11] (not a paper that is concerned with refinement – but that is not the issue) has state components that represent the current state of an editor display. Implicitly it is assumed that these are in some way "visible", even though there are no operations which observe it. In multi-language specification frameworks (e.g. ODP [10] or the various combinations of Z and a behavioural notation [12, 18, 16, 5]), Z is often used as an "information viewpoint" language (in ODP terminology), which only describes the data types present in the system, possibly with the operations on them. Other notations are then used for describing the actual sequencing of operations. In such a set-up, the data type representation used in the Z sublanguage cannot always assumed to be hidden. As a consequence, due to assumed visibility of state components, the standard data refinement rules only apply to a limited extent. This means that systems of this type cannot be developed stepwise using the standard rules: the state variables which are (implicitly) designated to be visible may not be removed in data refinement steps. Refinements of such specifications should be "grey box" refinements, in which it is assumed that certain state components remain present throughout.

This paper illustrates how specifications in which certain state components are assumed to be visible (and possibly modifiable), so-called "grey box data types", can be interpreted in terms of the standard black box data types. From this interpretation we derive simple data refinement rules for grey box data types. These rules contain the restriction that the observable state components must remain part of the state. This restriction disappears when adapting a more general approach, viz. that of *displays* or *views*, which are operations which *indirectly* observe the state of a data type. However, in that approach, defining *modifiable* displays is an instance of the well-known view update problem.

The notation used in this paper will be Z, but analogous constructions can be given for other model-based specification languages.

The next section will define the notion of a grey box data type. Section 3 will then present data refinement rules for such types. In Sect. 4 we will define a variation of such types, called "display box" data types, and define refinement rules for those. The final section contains our conclusions.

2 Black Boxes and Grey Boxes

The assumed specification style in model-oriented languages like Z is the black box abstract data type style, commonly known as "states-and-operations", with the familiar refinement relation on it (cf. Appendix A). A black box data type is specified by a tuple ($State$,Ops,$Init$) in which $State$ is a definition of a state space, consisting of a collection of typed variables and a predicate[1] on those variables. Ops is a set of operations, each of which is specified as a relation between the state before, the state after, and possibly inputs and outputs. $Init$ is the initialisation: a satisfiable predicate on the state variables describing the possible initial states of the type. The more abstract description of refinement on which the refinement rules for e.g. Z are based [8, 9] contains besides an initialisation also a *finalisation*, which relates the final state of the abstract data type to the "global state". This finalisation is ignored in most presentations of Z refinement (recent work by Woodcock et al [19] being a notable exception). Its use is to report information from the "run" of the system back to the global state when the system terminates.

In actual use of Z, especially when systems are modelled which are not wholly within a computer, one often deviates from the strict black box approach.

Example 1 If we presented the following state schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;CM\;}} \\
money : \mathbb{N} \\
display : String \\
\hline
money = 0 \Rightarrow display = \texttt{"Insert 35p then press Coffee"} \\
money \geq 35 \Rightarrow display = \texttt{"Press Coffee"} \\
0 < money < 35 \Rightarrow display = \texttt{"Insert"} \frown (shownum\,(35 - money)) \\
\qquad\qquad\qquad\qquad\qquad \frown \texttt{"p more then press Coffee"} \\
\hline
\end{array}$$

(where $shownum$ is an assumed function for turning integers into strings) it would be immediately clear that $display$ represents a part of the state which is intended to be observable. Hardly anyone would object to the coffee machine internally maintaining its balance in Eurocents, but the displayed text could not be changed to French without causing customer complaints. □

The grey box approach to data types aims to make such distinctions explicit, and to provide safe refinement rules for specifications like the above. Because we want to model both the specification language notion of "observability" and the programming language based notion of "visibility" (i.e. observability plus modifiability), the state of a grey box data type should in general be partitioned into three parts: readable components, modifiable components, and private components. The meaning of modifying a component is more complicated than in a programming language, because states of grey box data types have predicates on their components which need to be preserved. Thus, we need to ask: when

[1] We will call this predicate the *state predicate* and not the *state invariant*, because the state predicate is an invariant but not necessarily the strongest invariant that actually holds.

should modification be allowed, and what effect should it have on other components which are linked to it by the state predicate? This is an instance of the well-known framing problem discussed in e.g. [3, 11]. In terms of the refinement calculus [14], the question we need to ask is: what is the frame F in our desired specification $F{:}[p\ ,(x' = x? \wedge p)]$ where p is the state predicate? Our choice is the following (possibly rather arbitrary): non-modifiable variables cannot be changed indirectly, and modifiable components only explicitly (i.e., the frame F contains only the variable x itself). In order to allow modifiable components which are "linked" by the predicate to be changed together, we allow simultaneous changes. Thus, we assume the following principle:

> *The values of a collection of modifiable components can at any time be changed to values such that the state predicate is maintained by leaving all other components unchanged.*

This is not an ideal solution, ideally one would want to specify that a change in one component should induce a *minimal* change in the other components – certainly private components should be allowed to be affected. However, any specification of "minimal change" would become unwieldy. The solution above satisfies at least two desirable properties: it induces no restrictions if the state predicate is true, and it results in moderately simple specifications and refinement rules further on.

To properly express grey box data types, we have to introduce some Z specifics, which we will assume are familiar to the reader – from now on state spaces, operations, initialisations, etc. are *schemas*. Let a subspace of a schema S be any schema A such that $A \Leftrightarrow S \upharpoonright A$ (the operation $\upharpoonright$ denotes projection of a schema onto the components of another one – we will sometimes use a bracketed list of variables for its second argument; projection is defined as existential quantification over the "other" components). Two schemas are disjoint if they have no common components.

Definition 1 (Grey box) A grey box data type is a tuple $(State, Read, RW, Ops, Init)$ such that $(State, Ops, Init)$ is a black box data type (called the *underlying* black box), and $Read$ and RW are disjoint subspaces of $State$, denoting the read-only components and the modifiable (read-write) components. □

Example 2 A grey box specification using our earlier coffee machine state schema, making explicit that *display* is observable and *money* is not, is $(CM, CM \upharpoonright (display), [\,], \{Coin, Coffee\}, Init)$ where

$CoinValues = \{1,2,5,10,20,50,100\}$

$$\begin{array}{|l}\hline \textit{Coin} \\ \Delta CM \\ coin? : CoinValues \\ \hline money' = money + coin? \\ \hline \end{array}$$

$$\begin{array}{|l}\hline \textit{Init} \\ CM' \\ \hline money' = 0 \\ \hline \end{array}$$

$$\begin{array}{|l}\hline \textit{Coffee} \\ \Delta CM \\ \hline money' = money - 35 \\ \hline \end{array}$$

The read-only components are given by the projection of CM to the component *display*, which is a schema containing *display* as its only component and as its predicate that *display* has one of the values that CM might assign to it. Thus, *display* is always observable.

[ ] represents the empty schema, so there are no modifiable components in this example. It would not make sense to attempt to make *money* modifiable, because modifying *money* would only be allowed when that had no effect on *display*, e.g. when $money \geq 35$. Besides, it would allow changing the balance by an arbitrary amount, even a negative one. □

A grey box data type $(State, Read, RW, Ops, Init)$ can be interpreted as a black box data type based on the underlying one, which has an extended set of operations: observing operations for every component in $Read \wedge RW$, and a simultaneous modification operation for all components in RW.

Definition 2 (Interpretation of a grey box) The black box interpretation of a grey box $(State, Read, RW, Ops, Init)$ is $(State, Ops \cup \{Mod\} \cup Obs, Init)$ where Obs contains for every component $x : T$ of $Read \wedge RW$ the operation

$$\begin{array}{|l}\hline \textit{Obsx} \\ \Xi State \\ x! : T \\ \hline x! = x \\ \hline \end{array}$$

Let the components of RW be $x_i : T_i$ $(i = 1 \mathrel{..} n)$, then Mod is given by

$$\begin{array}{|l}\hline \textit{Mod} \\ \Delta RW \\ \Xi(State \setminus RW) \\ x_i? : T_i \; (i = 1 \mathrel{..} n) \\ \hline \forall\, i : 1 \mathrel{..} n \bullet x_i' = x_i? \\ \hline \end{array}$$

□

Note that this is not the only possible interpretation of a grey box as a black box: the observability of components could also be represented by extending the underlying black box by a finalisation. We will further discuss this in Sect. 5.

Example 3 In the coffee machine (Example 2) the interpreting black box contains as the only extra operation:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Obsdisplay}} \\
\hline
\Xi CM \\
display! : String \\
\hline
display! = display \\
\hline
\end{array}$$

This corresponds to our initial intuition that *display* really was observable. Now this is made explicit by the operation *Obsdisplay* which can always be performed and outputs the observable value. □

Example 4 An (imaginary) ancient machine for displaying four bit numbers has four switches, a handle, and a display. When you turn the handle, the display changes to the number represented by the current setting of the switches. This is specified by the grey box $(Anc, Anc \upharpoonright (disp), Anc \setminus (disp), \{Handle\}, Init)$ where

$bit = \{0,1\}$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Anc}} \\
\hline
sw1, sw2, sw4, sw8 : bit \\
disp : \mathbb{N} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Init}} \\
\hline
Anc' \\
\hline
disp' = 0 \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Handle}} \\
\hline
\Delta Anc \\
\hline
\Xi(Anc \setminus (disp)) \\
disp' = sw1 + 2*sw2 + 4*sw4 + 8*sw8 \\
\hline
\end{array}$$

The interpretation as a grey box contains observation operations

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Obsdisp}} \\
\hline
\Xi Anc \\
disp! : \mathbb{N} \\
\hline
disp! = disp \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Obssw1}} \\
\hline
\Xi Anc \\
sw1! : bit \\
\hline
sw1! = sw1 \\
\hline
\end{array}$$

(and similarly for the other switches), plus a modification operation

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Mod}} \\
\hline
\Delta Anc \\
sw1?, sw2?, sw4?, sw8? : bit \\
\hline
disp' = disp \\
sw1' = sw1? \land sw2' = sw2? \land sw4' = sw4? \land sw8' = sw8? \\
\hline
\end{array}$$

□

3 Refinement of Grey Box Data Types

To use grey box data types in developments, we need to define a refinement relation for them. Clearly we could take the approach that all grey boxes need to be replaced by their interpreting black boxes, and do refinement on those. This is always a possibility, however we would like to be able to stay within the grey box domain as long as possible. The grey box refinement relation will be based on refinement of their interpreting black boxes: two grey boxes are in the grey box refinement relation when the black boxes that interpret them are in the standard black box refinement relation.

Operation refinement (i.e., refinement in which the state space does not change) of grey boxes is not a very interesting issue. The rules for the operations are just the same as for black box operation refinement, which follows from the fact that every operation in the grey box becomes an operation in the black box and the fact that the rules for operation refinement are really independent between the various operations. In the interpretation as black boxes, non-trivial operation refinement of operations $Obsx$ is not possible because these operations are already total and deterministic. Operation refinements of the modification operation Mod are possible, however they will not normally result in black boxes that represent grey boxes. (But they could implement some of the more sophisticated methods for modification of linked variables, which is reassuring.)

Example 5 Consider the grey box $(S, S \upharpoonright (x), [\,], \varnothing, Init)$ where

$$\begin{array}{|l}
\multicolumn{1}{l}{S} \\ \hline
x, y, z : \mathbb{N} \\ \hline
y = x + z \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{Init} \\ \hline
S' \\ \hline
x = y = 0 \\ \hline
\end{array}$$

Even if it does define x to be modifiable, its modification operation Mod is very limited because it only allows x to change when that incurs no change in y or z, i.e. when x "changes" to its current value. However, one could imagine a more sophisticated modification operation on x which leaves z unchanged and changes y accordingly

$$\begin{array}{|l}
\multicolumn{1}{l}{Mod2} \\ \hline
\Delta S \\
x? : \mathbb{N} \\ \hline
x' = x? \\
z' = z \\ \hline
\end{array}$$

which is an operation refinement of Mod (but no longer the modify operation of any grey box data type). $\square$

In data refinement of grey box interpretations, we cannot change any of the observable state components. This fits with the interpretation of grey box data refinement as inheritance, however we will see how it can be removed in a later section. It follows from two issues: first, the type of $x!$ in $Obsx$ cannot change in data refinement, because inputs and outputs are not changed in data refinement. Second, the predicate of $Obsx$ can change in data refinement, but when it no longer has the shape $x = x!$ it is no longer an observation operation introduced in the black box interpretation of a grey box. Thus, $Obsx$ will have to keep variable x, and as a consequence so will the state. Thus, data refinement between two grey boxes will in the most general case be between $(AS, Read, RW, AOps, AI)$ and $(CS, Read, RW, COps, CI)$, using a *retrieve relation* R whose signature is $AS \wedge CS$ (with AS and CS sharing all but their private components). The rules for initialisation and between $AOps$ and $COps$ will be the same as those for the underlying black box. For completeness, these have been included in Appendix A. Now we need to investigate what refinement conditions derive from the implicit operations.

Observation operations The precondition of any observation operation is true for all possible states. Thus, the applicability condition for observation operations reduces to true. The correctness condition for observation operations also reduces to true because the state is unchanged and the output equals a component that is unchanged in data refinement.

Modification operation The analysis for the modification operation is slightly more complicated. Two crucial observations are that it is a *deterministic* operation, whose precondition is that its after-state is allowed.

Definition 3 For schemas A and B, the schema $A?_B$ denotes the schema obtained from A by decorating every component from B with a "?". Also, $A?_A$ gets abbreviated to $A?$. □

Using this convention, $\mathrm{pre}\, Mod = State?_{RW}$. Applicability then becomes $AS?_{RW} \wedge R \Rightarrow CS?_{RW}$, and correctness between $AMod$ and $CMod$ becomes $R \wedge AMod \wedge CMod \Rightarrow R'$ – informally, changing the same modifiable variables to the same values in two linked states should result in linked states afterwards.

Definition 4 (Grey box data refinement) The grey box data type $(CS, Read, RW, COps, CI)$ is a data refinement of $(AS, Read, RW, AOps, AI)$ when there exists a retrieve relation R whose signature is $AS \wedge CS$ such that

underlying black boxes $(CS, COps, CI)$ is a black box data refinement of $(AS, AOps, AI)$ using retrieve relation R (cf. Appendix A).

modifiability Any modification in the concrete type is possible when it is possible in the abstract type:

$\forall AS; CS; RW? \bullet AS?_{RW} \wedge R \Rightarrow CS?_{RW}$

correct modification For $AMod$ and $CMod$ the modification operations of the two types (cf. Definition 2):

$\forall AS; CS; AS'; CS'; RW? \bullet R \wedge AMod \wedge CMod \Rightarrow R'$ □

First we will present (contrived) examples of data refinements that fail to hold due to either of the grey box specific conditions, these demonstrate that the new conditions are independent. Then we will give one example of correct data refinement.

Example 6 The grey box data type $(CS,[\,],[x{:}\mathbb{N}],COps,CI)$ is not a data refinement of $(AS,[\,],[x{:}\mathbb{N}],AOps,AI)$ for the given retrieve relation R:

$$\begin{array}{|l}\hline AS \\ x,y:\mathbb{N} \\ \hline x=y \lor x=y+1 \\ \hline\end{array} \qquad \begin{array}{|l}\hline CS \\ x,z:\mathbb{N} \\ \hline x=z \\ \hline\end{array} \qquad \begin{array}{|l}\hline R \\ AS \\ CS \\ \hline y=z \\ \hline\end{array}$$

It fails on the modifiability condition, because in the state where $x{=}y{=}2$ in AS, x may be modified to 3, whereas in the corresponding state in CS, i.e. where $x{=}z{=}2$, it may not. In terms of the interpretation, the modification operation allows $x?{=}3$ in the first but not in the second. □

Example 7 Consider the grey boxes $(S, S \restriction (x), [\,], Ops, InS)$ and $(T, T \restriction (x), [\,], Ops, InT)$

$$\begin{array}{|l}\hline S \\ x,y:\mathbb{N} \\ \hline x=y \lor x=y+1 \\ \hline\end{array} \qquad \begin{array}{|l}\hline T \\ x,z:\mathbb{N} \\ \hline x=z \lor x+1=z \\ \hline\end{array}$$

$$\begin{array}{|l}\hline InS \\ S' \\ \hline x'=1 \\ \hline\end{array} \qquad \begin{array}{|l}\hline InT \\ T' \\ \hline x'=1 \\ \hline\end{array}$$

with retrieve relation $R \mathrel{\widehat=} [\, S;\ T \mid x{=}y \,]$. When $x, y{=}5$ in S, x may be modified to 6 (leaving y unchanged). A related state in T is $x{=}5, z{=}6$ and also here x may be modified to 6 leaving z unchanged. However, the resulting states are unrelated. Thus, in this case refinement fails on the condition of modification correctness. □

Example 8 A first attempt to extend the machine of Example 4 with negative numbers could be to add a third, "-1" position to all of the switches. This would result in the grey box $(Anc2, Anc2 \restriction (disp), Anc2 \setminus (disp), \{Handle\}, Init)$ where $bit{=}\{0, 1, -1\}$

$$\begin{array}{|l}\hline Anc2 \\ sw1, sw2, sw4, sw8 : bit \\ disp{:}\mathbb{Z} \\ \hline\end{array} \qquad \begin{array}{|l}\hline Init \\ Anc2' \\ \hline disp'=0 \\ \hline\end{array}$$

$$\begin{array}{|l}\hline Handle \\ \Delta Anc2 \\ \hline \Xi(Anc2 \setminus (disp)) \\ disp' = sw1 + 2{*}sw2 + 4{*}sw4 + 8{*}sw8 \\ \hline\end{array}$$

Using *Anc* itself as the retrieve relation, the underlying black boxes are clearly related by data refinement. Both modification operations are total, so the modifiability condition is satisfied. Correct modification follows from the fact that the concrete modification operation coincides with the abstract one on their common domain, and that the retrieve relation is the identity on that domain. (We do not need to consider the observation operations in the interpretations at all because they will be refinements thanks to the grey box formalisation.) □

The data refinement rule given above for grey boxes is not complete. Clearly a grey box which has an *explicit* operation of the form $Obsx \mathrel{\widehat=} [\,\Xi S\,;\ x!:T \mid x! = x\,]$ for a private variable x is equivalent (in the interpretation as a black box, at least) to the grey box which has included x in the observable variables instead. Thus, for a complete refinement rule one should consider the interpreting black boxes rather than the grey boxes – but we cannot think of any examples of incompleteness which are less artificial than the one given.

A more serious issue with grey box refinement is that it requires, throughout stepwise development, every observed variable to remain present as a state variable, even if the information that is to be "observed" could also be constructed from the state in another way. This problem is overcome by using so-called display boxes instead.

4 Display Boxes

A variant on the grey box data type is the "display box" data type, which has no directly observable components and no modifiable components, but explicit "observations" or displays. These observations relate the state to some output type, and should be total, i.e. there is no situation in which the observable aspect of the state can *not* be observed.

Example 9 The state space of our initial coffee machine could be cleaned up by separating out the *display* field and having that as a display instead:

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{CM2}} \\ \hline money : \mathbb{N} \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{Display}} \\ \hline CM2 \\ display! : String \\ \hline money = 0 \Rightarrow display! = \texttt{"Insert 35p then press Coffee"} \\ money \geq 35 \Rightarrow display! = \texttt{"Press Coffee"} \\ 0 < money < 35 \Rightarrow display! = \texttt{"Insert"} \frown (shownum\,(35 - money)) \\ \qquad\qquad\qquad\qquad \frown \texttt{"p more then press Coffee"} \\ \hline \end{array}$$

□

Definition 5 (Display box) A display box data type is a tuple $(State,Ds,Ops,Init)$ such that $(State,Ops,Init)$ is a black box data type, and every element D of the set Ds is a schema on $State$ and some other ("output") type, such that D is total, i.e. $D \upharpoonright State = State$. □

The informal interpretation of a display D is that it gives an output for every possible state. If each of the displays outputs the value of one state variable, the display box has the same interpretation as a grey box without modifiable variables. (Modifiable displays form an instance of the view update problem, and will be discussed later on.) The interpretation as a black box is very close to the display box: it just involves making $\Xi State$ explicit in every display.

Definition 6 (Interpretation of a display box) The display box $(State,Ds,Ops,Init)$ is interpreted as the black box $(State, Ops \cup Disps, Init)$ where $Disps$ contains for every element D of Ds the operation $D \wedge \Xi State$. □

Example 10 The coffee machine could be specified as the display box $(CM2,\{Display\},\{Coin,Coffee\},Init)$ and its interpretation would be the black box $(CM2,\{Coin,Coffee,(Display \wedge \Xi CM2)\},Init)$. (The only specification freedom that is lost by turning a state component into a display is the possibility to specify, for a non-functional display, which of the possible display values is to be chosen in the initial state.) □

As in the case of grey boxes, we need to define refinement for display boxes, by translating back refinements of interpreting black boxes. For this purpose, we will employ the technique of *calculating* most general data refinements [19, 2]. The correctness and applicability conditions for the most general data refinement of a display operation reduce to true.

Definition 7 (Display box refinement) The display box $(AS,ADs,AOps,AI)$ is data refined by display box $(CS,CDs,COps,CI)$ using retrieve relation R with signature $AS \wedge CS$ if

underlying black box $(AS,AOps,AI)$ is data refined by $(CS,COps,CI)$ using retrieve relation R.

displays The displays in ADs and CDs can be matched in pairs AD,CD such that CD is an operation refinement of $(\exists AS \bullet AD \wedge R) \vee \neg (\exists AS \bullet R)$. □

The calculated most general data refinement of AD is actually $(\exists AS \bullet AD \wedge R)$, however, this is not a total operation, which is required for displays. The given expression is the most general total data refinement of that, totalising it by allowing any display for states that are unrelated by the retrieve relation.

Example 11 Having redefined the coffee machine of Example 2 as a display box in Example 3, we can now present the internal adaptation of the European common currency as a display box refinement. With retrieve relation

$$
\begin{array}{|l}
\hline
\textit{Exchange} \\
money, geld : \mathbb{N} \\
\hline
geld = 3 * money \\
\hline
\end{array}
$$

we have that the display box $(CM2, \{Display\}, \{Coin, Coffee\}, Init)$ is data refined by $(KM, \{Zeige\}, \{Muenze, Kaffee\}, Anfang)$ where

$CoinValues = \{1,2,5,10,20,50,100\}$

$$
\begin{array}{|l}
\hline
\textit{Muenze} \\
\Delta KM \\
coin? : CoinValues \\
\hline
geld' = geld + coin? * 3 \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{Anfang} \\
KM' \\
\hline
geld' = 0 \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{Kaffee} \\
\Delta KM \\
\hline
geld' = geld - 105 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Zeige} \\
KM \\
display! : String \\
\hline
geld = 0 \Rightarrow display! = \texttt{"Insert 35p then press Coffee"} \\
geld \geq 105 \Rightarrow display! = \texttt{"Press Coffee"} \\
0 < geld < 105 \Rightarrow display! = \texttt{"Insert"} \frown (shownum\,((105 - geld) \operatorname{div} 3)) \\
\qquad\qquad\qquad\qquad \frown \texttt{"p more then press Coffee"} \\
\hline
\end{array}
$$

The calculated data refinement for $Display$ would leave $display!$ unspecified when $geld$ is not divisible by 3, the operation $Zeige$ is the (syntactically) simplest deterministic operation refinement of that.

The link to the grey box example of the same coffee machine is also given by a display box refinement. If we refine $CW2$ to CW using CW as the retrieve relation, this introduces $display$ as a state component with the obvious value, then the calculated new display will be

$$
\begin{array}{|l}
\hline
\textit{NewDisp} \\
CW2 \\
display! : String \\
\hline
display! = display \\
\hline
\end{array}
$$

whose black box interpretation is of course identical to the implicit operation $Obsdisplay$ in the grey box. □

Display boxes have the advantage over grey boxes that they allow indirect observations of variables, which in turn allow a broader range of data refinements.

However, there is also a downside to using display boxes: defining modifiable displays is problematic. This is very similar to the well-known and extensively studied view update problem in databases [1], and to linking displays and updates in visualisation systems [13, 15]. Displays are defined in terms of state variables, but it is usually not clear how an explicit change in a display should be translated back to changes in those variables.

Example 12 Given the display box data type $(WCFinal, \{Voor, ByShearer\}, \{Doelpunt, Goal\}, KickOff)$ where

$$team ::= Engl \mid Holl$$

$$\begin{array}{|l}\hline WCFinal \\ goals : team \to \mathbb{N} \\ \hline \end{array}$$

$$\begin{array}{|l}\hline KickOff \\ WCFinal' \\ \hline goals'\ Holl = 0 \\ goals'\ Engl = 0 \\ \hline \end{array}$$

$$\begin{array}{|l}\hline ByShearer \\ WCFinal \\ scored! : \mathbb{N} \\ \hline scored! = goals\ Engl \\ \hline \end{array}$$

$$\begin{array}{|l}\hline Doelpunt \\ \Delta WCFinal \\ \hline goals'\ Holl = goals\ Holl + 1 \\ goals'\ Engl = goals\ Engl \\ \hline \end{array}$$

$$\begin{array}{|l}\hline Goal \\ \Delta WCFinal \\ \hline goals'\ Engl = goals\ Engl + 1 \\ goals'\ Holl = goals\ Holl \\ \hline \end{array}$$

$$\begin{array}{|l}\hline Voor \\ WCFinal \\ lead! : \mathbb{Z} \\ \hline lead! = goals\ Holl - goals\ Engl \\ \hline \end{array}$$

we could not make any of these displays modifiable: it is impossible to determine the number of goals scored by either side from the difference between the two, or from the number of goals scored by one side only. As it happens, from both displays together we can draw enough information, but in general even this need not be the case. $\square$

Another example of a display that could not be made modifiable is the one in the coffee machine: how much money is in the machine when the display reads `"Press Coffee"`?

We could introduce a data type with updateable displays by introducing the restriction that updateable displays are *injective*. However, this would result in seriously constrained data refinement rules, and thus we have omitted this alternative.

5 Summary and Conclusions

We have defined the concepts of grey box and display box data types, by giving interpretations of these in terms of the traditional black box data types. By use of many examples we have shown that our alternative types can be used to simplify specifications, and to formalise informal styles of specification and development which assume that certain state components are hidden. In particular, we have given refinement rules which operate on grey boxes and display boxes directly, whose soundness follows from black box refinement rules between their interpretations. The derived refinement rules were considerably simplified from the original black box interpretation refinement conditions, due to the extra structure of the specifications. In particular, observability of variables in grey boxes is defined in a way which ensures that it imposes no conditions on data refinement. Grey boxes have the advantage that they may include modifiable variables, and the disadvantage that only limited forms of data refinement (viz. those that change only private variables) are possible. In display boxes, the latter disadvantage disappears, at the price of losing the option for implicit modification. However, since both are defined in terms of their underlying black boxes with extra operations added to them, a mixture of grey box and display box data types seems well possible.

The paper has left unexplored the possibility of defining observable variables and displays in terms of a *finalisation*. The standard presentation of black box data types has an empty finalisation, which means that the only way for the system to communicate values to its environment is by the outputs of operations. In the most general model [8] this communication also (or *only*) happens after the system has completed its "run" in a finalisation step. Any state component (for observable variables) or expression in terms of state components (for displays) that is included as a system output in the finalisation has to be viewed as "observable", because finalisation may happen at any time. Thus, in interpreting grey and display boxes as black boxes, we could have included displays and observation of variables in a finalisation rather than in new operations. This may be somewhat "cleaner" although semantically there should be no difference, because the rules for refinement between operations with outputs are derived from those for systems where all output occurs at finalisation [19].

References

1. E. Bertino and G. Guerrini. Viewpoints in object database systems. In A. Finkelstein and G. Spanoudakis, editors, *SIGSOFT '96 International Workshop on Multiple Perspectives in Software Development (Viewpoints '96)*, pages 289–293. ACM, 1996.
2. E.A. Boiten, J. Derrick, H. Bowman, and M. Steen. Coupling schemas: data refinement and view(point) composition. In D.J. Duke and A.S. Evans, editors, *Northern Formal Methods Workshop*, Electronic Workshops In Computing. Springer, 1997.
3. A. Borgida, J. Mylopoulos, and R. Reiter. And nothing else changes: The frame problem in procedure specifications. In *Proc. 15th International Conference on Software Engineering*, Baltimore, Maryland, May 1993. IEEE Computer Society Press.
4. K. Bruce, L. Cardelli, G. Castagna, The Hopkins Object Group, G.T. Leavens, and B. Pierce. On binary methods. *Theory and Practice of Object Systems*, 1(3):221–242, 1996.

5. C. Fischer. CSP-OZ: a combination of Object-Z and CSP. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-based Distributed Systems*, volume 2, pages 423–438. Chapman & Hall, 1997.
6. M.-C. Gaudel and J. Woodcock, editors. *FME'96: Industrial Benefit of Formal Methods, Third International Symposium of Formal Methods Europe*, LNCS 1051. Springer-Verlag, March 1996.
7. A. Goldberg. *Smalltalk-80 — The language and its implementation.* Addison-Wesley, 1983.
8. He Jifeng, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP 86*, LNCS 213, pages 187–196. Springer-Verlag, 1986.
9. He Jifeng and C.A.R. Hoare. Prespecification and data refinement. In *Data Refinement in a Categorical Setting*, Technical Monograph PRG-90. Oxford University Computing Laboratory, November 1990.
10. ITU Recommendation X.901-904 — ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model - Parts 1-4*, July 1995.
11. D. Jackson. Structuring Z specifications with views. *ACM Transactions on Software Engineering and Methodology*, 4(4), October 1995.
12. V. Kasurinen and K. Sere. Integrating action systems and Z in a medical system specification. In FME'96 [6], pages 105–119.
13. G.J. Klinker. An environment for telecollaborative data exploration. In *Proceedings Visualization '93 – sponsored by the IEEE Computer Society*, pages 110–117, 1993.
14. C. C. Morgan. *Programming from Specifications.* Prentice Hall International Series in Computer Science, 2nd edition, 1994.
15. J.C. Roberts. On encouraging multiple views for visualization. In *Information Visualization IV'98*, London, July 1998. IEEE Computer Society.
16. G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *FME'97: Industrial Application and Strengthened Foundations of Formal Methods*, LNCS 1313, pages 62–81. Springer-Verlag, September 1997.
17. J. M. Spivey. *The Z notation: A reference manual.* Prentice Hall, 1989.
18. M. Weber. Combining statecharts and Z for the design of safety-critical control systems. In FME'96 [6], pages 307–326.
19. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall, 1996.

A Data Refinement Rules for Black Boxes

Given black box data types A=(AS,$AOps$,$AInit$) and C=(CS,$COps$,$CInit$), then C is a data refinement of A if[2] using retrieve relation R (whose signature is $AS \wedge CS$) if the following conditions hold:

initialisation $\forall\, CS \bullet CInit \Rightarrow (\exists\, AS \bullet AInit \wedge R)$

[2] "if" but not "iff", these are the conditions for *forwards* simulation, which are only sufficient for data refinement in combination with those for *backwards* simulation (cf. [8, 19]). However, operations introduced in this paper are deterministic, for which case forwards simulation is sufficient.

and the operations in $AOps$ and $COps$ can be matched in pairs AOp,COp both with input $x? : X$ and output $y! : Y$, such that for each of those pairs the following two conditions hold:

applicability COp should be defined on all representatives of AS on which AOp is defined:

$\forall AS\ ;\ CS\ ;\ x? : X \bullet \mathrm{pre}\, AOp \land R \ \Rightarrow\ \mathrm{pre}\, COp$

correctness wherever AOp is defined, COp should produce a result related by R to one that AOp could have produced:

$$\begin{array}{l}\forall AS\ ;\ CS\ ;\ CS'\ ;\ x? : X\ ;\ y! : Y \bullet \\ \quad \mathrm{pre}\, AOp \land COp \land R \ \Rightarrow\ \exists AS' \bullet R' \land AOp\end{array}$$