Citation for published version


DOI

Link to record in KAR

https://kar.kent.ac.uk/21607/

Document Version

UNSPECIFIED

Copyright & reuse
Content in the Kent Academic Repository is made available for research purposes. Unless otherwise stated all content is protected by copyright and in the absence of an open licence (e.g., Creative Commons), permissions for further reuse of content should be sought from the publisher, author or other copyright holder.

Versions of research
The version in the Kent Academic Repository may differ from the final published version. Users are advised to check http://kar.kent.ac.uk for the status of the paper. Users should always cite the published version of record.

Enquiries
For any further enquiries regarding the licence status of this document, please contact:

researchsupport@kent.ac.uk

If you believe this document infringes copyright then please contact the KAR admin team with the take-down information provided at http://kar.kent.ac.uk/contact.html
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 implementation. 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 \((\text{State}, \text{Ops}, \text{Init})\) in which \text{State} is a definition of a state space, consisting of a collection of typed variables and a predicate\(^1\) on those variables. \text{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. \text{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 \textit{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.

\textbf{Example 1}  \quad If we presented the following state schema:

\begin{verbatim}
CM
money:N
display:String

money=0 \Rightarrow display="Insert 35p then press Coffee"
money \geq 35 \Rightarrow display="Press Coffee"
0<money<35 \Rightarrow display = "Insert" \land (shownum (35-money))
\land "p more then press Coffee"
\end{verbatim}

(where \textit{shownum} is an assumed function for turning integers into strings) it would be immediately clear that \textit{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. \hfill \square

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 \textit{state predicate} and not the \textit{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 $F;[p,(x' = x \land p)]$ 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 $(\text{State}, \text{Read}, \text{RW}, \text{Ops}, \text{Init})$ such that $(\text{State}, \text{Ops}, \text{Init})$ 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 $(\text{CM}, \text{CM} \upharpoonright (\text{display}), [], \text{\{Coin,Coffee\},Init})$ where $\text{CoinValues}=\{1,2,5,10,20,50,100\}$
The read-only components are given by the projection of $CM$ to the component $\text{display}$, which is a schema containing $\text{display}$ as its only component and as its predicate that $\text{display}$ has one of the values that $CM$ might assign to it. Thus, $\text{display}$ 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 $\text{money}$ modifiable, because modifying $\text{money}$ would only be allowed when that had no effect on $\text{display}$, e.g. when $\text{money} \geq 35$. Besides, it would allow changing the balance by an arbitrary amount, even a negative one.

A grey box data type $(\text{State, Read, RW, Ops, Init})$ 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 $\text{Read} \land \text{RW}$, and a simultaneous modification operation for all components in $\text{RW}$.

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

\[
\text{Obs}_x = \begin{cases} 
\text{State} \\
 x!: T \\
 x! = x
\end{cases}
\]

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

\[
\text{Mod} = \begin{cases} 
\text{RW} \\
 \text{Mod}_{\text{RW}} \\
 \text{Obs}_x \ \text{for } x \in \text{RW} \\
 \forall i : 1 \ldots n \bullet x'_i = x_i
\end{cases}
\]

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:

\[
\begin{align*}
\text{Obsdisplay} & \equiv CM \\
\text{display} : String \\
\text{display} = \text{display}
\end{align*}
\]

This corresponds to our initial intuition that \text{display} really was observable. Now this is made explicit by the operation \text{Obsdisplay} which can always be performed and outputs the observable value. \hfill \Box

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 \((\text{Anc}, \text{Anc} \mid (\text{disp}), \text{Anc} \setminus (\text{disp}), \{\text{Handle}\}, \text{Init})\) where

\[
\begin{align*}
\text{bit} & = \{0,1\} \\
\text{Anc} & = \text{sw}1, \text{sw}2, \text{sw}4, \text{sw}8 : \text{bit} \\
\text{disp} & \equiv \text{N} \\
\text{Anc} & \equiv \text{disp} \\
\text{Handle} & \equiv \Delta \text{Anc} \\
\text{disp}' & = \text{sw}1 + 2 \times \text{sw}2 + 4 \times \text{sw}4 + 8 \times \text{sw}8
\end{align*}
\]

The interpretation as a grey box contains observation operations

\[
\begin{align*}
\text{Obsdisp} & \equiv \text{Anc} \\
\text{disp} : \text{N} \\
\text{disp} = \text{disp} \\
\text{Obssw1} & \equiv \text{Anc} \\
\text{sw}1 : \text{bit} \\
\text{sw}1 = \text{sw}1
\end{align*}
\]

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

\[
\begin{align*}
\text{Mod} & \equiv \Delta \text{Anc} \\
\text{sw}1, \text{sw}2, \text{sw}4, \text{sw}8 : \text{bit} \\
\text{disp}' & = \text{disp} \\
\text{sw}1' & = \text{sw}1' \land \text{sw}2' = \text{sw}2' \land \text{sw}4' = \text{sw}4' \land \text{sw}8' = \text{sw}8'
\end{align*}
\]

\hfill \Box
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 $(S, S | (x), [], @, \text{Init})$ where

$$
\begin{array}{c}
\text{S} \\
\text{x,y,z:N} \\
y = x + z
\end{array} \quad \begin{array}{c}
\text{Init} \\
\text{S'} \\
x = y = 0
\end{array}
$$

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

$$
\begin{array}{c}
\text{Mod2} \\
\Delta S \\
x?: N
\end{array} \quad \begin{array}{c}
x' = x? \\
z' = z
\end{array}
$$

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

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 $\texttt{Obsx}$ cannot change in
data refinement, because inputs and outputs are not changed in data refinement.
Second, the predicate of $\texttt{Obsx}$ 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, $\texttt{Obsx}$ 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 $(\texttt{AS,Read,RW,APs,A})$
and $(\texttt{CS,Read,RW,COps,C})$, using a retrieve relation $R$ whose signature is
$\texttt{AS\&CS}$ (with $\texttt{AS}$ and $\texttt{CS}$ sharing all but their private components). The
rules for initialisation and between $\texttt{APs}$ and $\texttt{COps}$ 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 oper-
ation, 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 \\
$\texttt{\_?\_}$. Also, $A?^A$ gets abbreviated to $A?_A$. □

Using this convention, $\texttt{pre Mod = State?_{RW}}$. Applicability then becomes
$\texttt{AS?_{RW} \land R} \Rightarrow \texttt{CS?_{RW}}$, and correctness between $\texttt{AMod}$ and $\texttt{CMod}$ becomes
$R \land \texttt{AMod} \land \texttt{CMod} \Rightarrow R'$ — 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
$(\texttt{CS,Read,RW,COps,C})$ is a data refinement of $(\texttt{AS,Read,RW,APs,A})$
when there exists a retrieve relation $R$ whose signature is $\texttt{AS\&CS}$ such that

underlying black boxes $(\texttt{CS,COps,C})$ is a black box data refinement of
$(\texttt{AS,APs,A})$ using retrieve relation $R$ (cf. Appendix A).
modifiability Any modification in the concrete type is possible when it is pos-
sible in the abstract type:
$\forall \texttt{AS, CS}; \texttt{RW}; \texttt{AS'}_{\texttt{ RW}} \land R \Rightarrow \texttt{CS'}_{\texttt{ RW}}$
correct modification For $\texttt{AMod}$ and $\texttt{CMod}$ the modification operations of
the two types (cf. Definition 2):
$\forall \texttt{AS, CS}; \texttt{AS'}; \texttt{CS'}; \texttt{RW}; \texttt{R} \land \texttt{AMod} \land \texttt{CMod} \Rightarrow R'$ □
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 \((CS, [], [x: N], COps, CI)\) is not a data refinement of \((AS, [], [x: N], AOps, AI)\) for the given retrieve relation \(R\):

\[
\begin{array}{c|c|c}
\text{AS} & CS & R \\
\hline
x, y: N & x, z: N & AS \\
\hline
x = y \lor x = y + 1 & x = z & CS \\
\hline
\end{array}
\]

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 \((S, S \mid (x), [], Ops, InS)\) and \((T, T \mid (x), [], Ops, InT)\) with retrieve relation \(R \triangleq [S; T \mid x = y]\). 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 \((Anc2, Anc2 \mid (disp), Anc2 \setminus (disp), \{Handle\}, Init)\) where \(bit = \{0, 1, -1\}\)

\[
\begin{array}{c|c|c}
\text{Anc2} & \text{Init} & \Xi (Anc2 \setminus (disp)) \\
\hline
\text{sw1}, \text{sw2}, \text{sw4}, \text{sw8}, \text{bit} & \text{Anc2'} & disp' = 0 \\
\hline
\text{disp}, \text{Z} & \Delta \text{Anc2} & disp' = sw1 + 2\times \text{sw2} + 4\times \text{sw4} + 8\times \text{sw8} \\
\hline
\end{array}
\]
Using \textit{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 \textit{explicit} operation of the form $\text{Obs} \mathbf{x} \triangleq \left[ \exists S ; \mathbf{x}! : T \ | \mathbf{x}! = \mathbf{x} \right]$ for a private variable $\mathbf{x}$ is equivalent (in the interpretation as a black box, at least) to the grey box which has included $\mathbf{x}$ 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 \textit{display} field and having that as a display instead:

\begin{verbatim}
<table>
<thead>
<tr>
<th>CM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>money:N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM2</td>
</tr>
<tr>
<td>display: String</td>
</tr>
</tbody>
</table>

money=0 \Rightarrow \text{display}="Insert 35p then press Coffee"

money \geq 35 \Rightarrow \text{display}="Press Coffee"

0<money<35 \Rightarrow \text{display}="Insert" \land (shownum (35-money))

\land "p more then press Coffee"
\end{verbatim}
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 $D | State = State.$

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 $\exists State$ explicit in every display.

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

Example 10 The coffee machine could be specified as the display box $(CM2, \{\text{Display}\}, \{\text{Coin, Coffee}\}, Init)$ and its interpretation would be the black box $(CM2, \{\text{Coin, Coffee, Display }\land \exists CM2\}, Init)$. (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 $(AS, ADs, Ops, AI)$ is data refined by display box $(CS, CDs, COps, CI)$ using retrieve relation $R$ with signature $AS \land CS$ if

underlying black box $(AS, Ops, AI)$ 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 $(\exists AS \cdot AD) \land (\exists AS \cdot R)$. □

The calculated most general data refinement of $AD$ is actually $(\exists AS \cdot AD \land R)$, 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
we have that the display box \( \{CM2,\{Display\},\{Coin,Coffee\},Init\} \) is data refined by \( \{KM,\{Zeige\},\{Muenze,Kaffee\},Anfang\} \) where

\[
\begin{align*}
\text{CoinValues} &= \{1,2,5,10,20,50,100\} \\
\text{Anfang} &\quad \text{KM'} \\
&\quad \text{geld' = 0} \\
\text{Kaffee} &\quad \Delta KM \\
&\quad \text{geld'} = \text{geld} - 105 \\
\end{align*}
\]

The calculated data refinement for \( \text{Display} \) would leave \( \text{display} \) unspecified when \( \text{geld} \) is not divisible by 3, the operation \( \text{Zeige} \) 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 \( \text{display} \) as a state component with the obvious value, then the calculated new display will be

\[
\begin{align*}
\text{NewDisp} &\quad CW2 \\
&\quad \text{display}': \text{String} \\
&\quad \text{display}' = \text{display} \\
\end{align*}
\]

whose black box interpretation is of course identical to the implicit operation \( \text{Obsdisplay} \) 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  
\((\text{WCFinal}, \{\text{Voor, ByShearer}\}, \{\text{Doelpunt, Goal}\}, \text{KickOff})\) where

\[
\begin{align*}
\text{team} & := \text{Engl} \mid \text{Hol}\l \\
\text{WCFinal} & \subseteq \text{goals: team} \rightarrow \mathbb{N} \\
\text{KickOff} & \subseteq \text{WCFinal} \\
\text{goals'} \text{Hol}\l & = 0 \\
\text{goals'} \text{Engl} & = 0 \\
\text{ByShearer} & \subseteq \text{WCFinal} \\
\text{scored} & \subseteq \mathbb{N} \\
\text{scored} & = \text{goals Engl}
\end{align*}
\]

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.

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 particu-
lar, 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 ex-
tra 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 modifica-
tion. 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. Finkel-
stein and G. Spanoudakis, editors, SIGSOFT '96 International Workshop on Mul-
tiple 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 re-
finemnt and view(point) composition. In D.J. Duke and A.S. Evans, editors, Northern
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
242, 1996.
A Data Refinement Rules for Black Boxes

Given black box data types $\mathbf{A}=(\mathbf{AS}, \mathbf{AOps}, \mathbf{AInit})$ and $\mathbf{C}=(\mathbf{CS}, \mathbf{COps}, \mathbf{CInit})$, then $\mathbf{C}$ is a data refinement of $\mathbf{A}$ if\footnote{“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. \cite{[8],[19]}). However, operations introduced in this paper are deterministic, for which case forwards simulation is sufficient.} using retrieve relation $R$ (whose signature is $\mathbf{AS} \times \mathbf{CS}$) if the following conditions hold:

initialisation $\forall \mathbf{CS} \bullet \mathbf{CInit} \Rightarrow (\exists \mathbf{AS} \bullet \mathbf{AInit} \land R)$
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:

\[ \forall AS; CS; x?: X \quad \text{pre} \; AOp \wedge R \Rightarrow \text{pre} \; COp \]

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

\[ \forall AS; CS; CS' ; x?: X; y!: Y \quad \text{pre} \; AOp \wedge COp \wedge R \Rightarrow \exists AS' \quad R' \wedge AOp \]