Exclusive Control within Object Oriented Systems

Jin Song Dong

Software Engineering Group
Division of Information Technology
CSIRO
Canberra, ACT 2601, Australia

Roger Duke

Software Verification Research Centre
Department of Computer Science
The University of Queensland
Brisbane, Q 4072, Australia

Abstract

In object-oriented systems, object associations generally result in a complex structure whose precise description is a crucial part of the system's formal specification. A particularly common object association arises when one object exclusively controls (owns) another. As exclusive object control is an important aspect of safety and security critical systems, specific notation to capture directly such a notion needs to be included in any formal specification language. This paper first discusses the problems that arise when using the existing Object-Z notations, such as object containment, to capture the notion of exclusive control. Then the Object-Z notation is extended to enable the notion of exclusive control to be modelled directly. Finally, a case study is presented to contrast the distinction between the use of exclusive control and object containment when modelling object-oriented systems.

Keywords: formal object-oriented specification, exclusive control, containment

1 Introduction

In object-oriented systems, a client object will have an attribute whose value references some other supplier object in the system so that the referenced supplier object can be sent messages (or accessed)[Meyer 1988]. The object associations determined by the attributes of client objects in a system will generally result in a complex structure. The precise description of these associations is a crucial part of the formal specification of the system. In this paper, we are interested in formally modelling a particular association, that of exclusive object control. The exclusive control notion arises when a supplier object has only one direct client object in a system. In this case, the client object has exclusive control over the supplier object. For instance, in a general case, a car object may be exclusively controlled by a person object. As the notion of exclusive object control is an important aspect of safety and security critical systems, there is the need for specific notations to capture directly such a notion within a formal specification language such as Object-Z[Duke, King, Rose, and Smith 1991; Duke, Rose, and Smith ; Rose 1992]. This paper extends the Object-Z notation to accommodate this need.

The notion of exclusive object control is at first sight apparently similar to the notion of object containment[Dong and Duke 1995a]. However, object containment is concerned only with the relative geometric patterns of some common object associations; it is not concerned with the access control aspect of object associations. Therefore, in general it is inappropriate to use the containment notations introduced in [Dong and Duke 1995a] to capture the notion of exclusive control. Section 2 of this paper details this with an example that motivates the need to extend Object-Z to capture directly the notion of exclusive control. Section 3 then presents the specific notation for exclusive object control and details the underlying semantics. Section 4 presents a case study to illustrate the role of exclusive control in system modelling. In the case study, exclusive control and containment notations are both used so that the distinction between the two can be demonstrated. Related

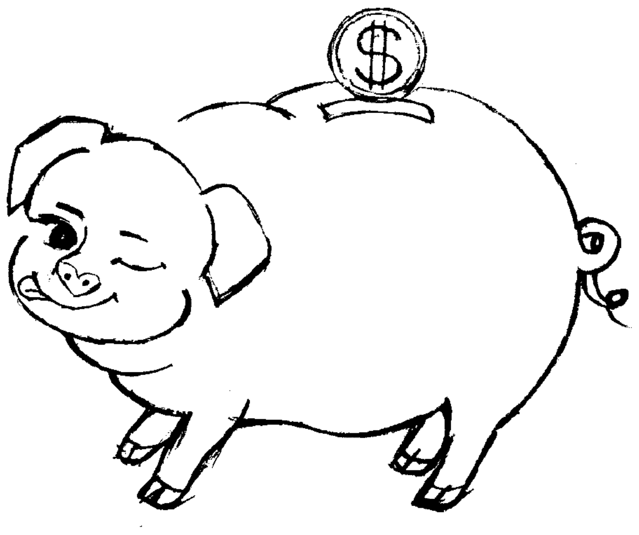

Figure 1: Piggy-bank.

issues, such as transferring object ownership and the notion of new objects, are also discussed in this case study.

Familiarity with the basic features of the Object-Z notations as given in [Duke, King, Rose, and Smith 1991; Rose 1992] is assumed. However, for those readers not familiar with object containment within Object-Z, a brief introduction is given in the Appendix.

2 Capturing Exclusive Control: the Problem

The reference semantics of Object-Z[Duke and Rose 1993; Rose 1992] takes the view that every referenced object is potentially shared. A declaration $a : A$ declares a to be a reference to an object of class A. There is no implication that distinct object reference declarations introduce distinct object references and hence distinct objects, i.e. declaration $a_1, a_2 : A$ does not imply that a_1 and a_2 reference distinct objects.

However, exclusive control supposes the existence of a unique client (the owner) with exclusive access. The notion of object containment as defined in [Dong and Duke 1995a] can ensure only that no object is directly (uniquely) *contained* in two distinct objects. It indicates nothing about whether objects other than a containing object can *reference* (and hence access) a contained object (The discussion in Section 5 of [Dong and Duke 1995a] clearly demonstrates that containment and access are distinct notions).

2.1 Example: Piggy-bank

Suppose each (adult) person has a piggy-bank for storing spare coins. Furthermore, suppose a married person shares their piggy-bank with their spouse, whereas a single person exclusively owns their piggy-bank. A piggy-bank can be modelled in Object-Z as:

```
┌─ PiggyBank ───────────────────────────
│
│  ┌──────────────────  ┌─ INIT ──────────
│  │ coins : ℙ Coins    │ coins = ∅
│  └──────────────────  └─────────────────
│
│  ┌─ PutIn ───────────────────────────
│  │ Δ(coins)
│  │ in? : Coin
│  ├──────────────
│  │ in? ∉ coins
│  │ coins′ = coins ∪ {in?}
│  └───────────────────────────────────
│
│  ┌─ EmptyAll ────────────────────────
│  │ Δ(coins)
│  │ out! : ℙ Coins
│  ├──────────────
│  │ out! = coins
│  │ coins′ = ∅
│  └───────────────────────────────────
└───────────────────────────────────────
```

where *Coins* denotes the set of all coins.

A specification of married and single people (with their piggy-banks) in Object-Z would be

MarriedPerson

$$\begin{array}{l} has_a : PiggyBank \\ spouse : MarriedPerson \\ \hline spouse \neq self \\ spouse.spouse = self \\ has_a = spouse.has_a \\ \forall\, m : (MarriedPerson \setminus \{self, spouse\}) \bullet \\ \qquad m.has_a \neq has_a \end{array}$$

$PutIn \mathrel{\widehat=} has_a.PutIn$

$EmptyAll \mathrel{\widehat=} has_a.EmptyAll$

SinglePerson

$has_a : PiggyBank_{©}$

$PutIn \mathrel{\widehat=} has_a.PutIn$

$EmptyAll \mathrel{\widehat=} has_a.EmptyAll$

Notice that the semantics of the containment '©' ensures the following property:

$$\begin{array}{l} \forall\, s_1, s_2 : SinglePerson \bullet \\ \qquad s_1 \neq s_2 \Rightarrow s_1.has_a \neq s_2.has_a \end{array}$$

which, however, by itself does not prevent an object of *SinglePerson* and an object of *MarriedPerson* referencing the same piggy-bank object. Moreover it is also inappropriate to use the object containment notation to specify the class *MarriedPerson*, such as $has_a : PiggyBank_{©}$ in the state schema of the class *MarriedPerson*, because the semantics of '©' will invalidate the property

$$\begin{array}{l} \forall\, m : MarriedPerson \bullet \\ \qquad m.has_a = m.spouse.has_a \end{array}$$

(the property which allows a married couple to share the same piggy-bank object.)

To ensure the exclusive control relation between single people and their piggy-banks, it is possible to further impose system constraints on the environment of those *SinglePerson* objects in a system, i.e.

System

$$\begin{array}{l} people : \mathbb{P}(SinglePerson \cup \\ \qquad MarriedPerson) \\ \hline \forall\, s : SinglePerson \bullet \qquad [\mathtt{i}] \\ \quad \forall\, m : MarriedPerson \bullet \\ \qquad s.has_a \neq m.has_a \end{array}$$

$PutIn \mathrel{\widehat=} [p? : people] \bullet p?.PutIn$

$EmptyAll \mathrel{\widehat=} [p? : people] \bullet p?.EmptyAll$

where $SinglePerson \cup MarriedPerson$ is a class-union (see [Dong ; Dong and Duke 1993] for details on class-union constructs). However, capturing the exclusive control property of the class *SinglePerson* explicitly in this way is cumbersome, particular if the specification is complex and contains many objects. Furthermore, if the specification needs to be extended to include a new class *Child*, say, which also has an attribute has_a referencing a piggy-bank object that may be shared with their parents, then the following predicate which is similar to [i] would need to be included in the specification.

$$\begin{array}{l} \forall\, s : SinglePerson \bullet \\ \qquad \forall\, c : Child \bullet s.has_a \neq c.has_a \end{array}$$

This problem indicates that we need a new notation to capture this exclusive control notion directly in the Object-Z specification language.

3 Notation for Exclusive Control

In this section, the properties of exclusive control are first formalised, and then notation is introduced into Object-Z to capture directly these properties.

3.1 Formalising Exclusive Control

Broadly speaking, if one object references another object, either the object exclusively controls the referenced objects and no other object can reference it, or the referenced object is free to be referenced by other objects.

To capture this idea formally, let $\mathbb{O}$ denote the universe of all object identities in a system (see [Dong and Duke 1993]), and let

$$ec : \mathbb{O} \leftrightarrow \mathbb{O}$$

denote the relation whereby

$$ob_1 \; \underline{ec} \; ob_2$$

if and only if object ob_1 exclusive controls object ob_2. In contrast, let

$$\mathit{free} : \mathbb{O} \leftrightarrow \mathbb{O}$$

denote the relation whereby

$$ob_1 \; \underline{\mathit{free}} \; ob_2$$

if and only if object ob_1 references ob_2, but ob_2 is free to be referenced by other objects.

The properties of the relations *ec* and *free* can be formally captured as

$$\begin{array}{ll} \forall\, ob : \mathbb{O} \bullet & \texttt{[ii]} \\ \quad ec(\llparenthesis \{ob\} \rrparenthesis) \cap \mathit{free}(\llparenthesis \{ob\} \rrparenthesis) = \varnothing & \\ \forall\, ob_1, ob_2 : \mathbb{O} \bullet & \texttt{[iii]} \\ \quad ob_1 \neq ob_2 \Rightarrow & \\ \qquad ec(\llparenthesis \{ob_1\} \rrparenthesis) \cap (ec \cup \mathit{free})(\llparenthesis \{ob_2\} \rrparenthesis) = \varnothing & \end{array}$$

3.2 Object-Z Notation for Exclusive Control

In this subsection, specific notation is introduced into Object-Z to capture directly the notion of exclusive object control. This notation enables the specifier to state explicitly, as part of the class specification, which objects will be exclusively controlled by objects of that class.

Let each class have implicitly declared secondary attributes[1]

$$\mathit{excon}, \mathit{freerefs} : \mathbb{P}\,\mathbb{O}$$

where the value of *excon* is the set of exclusive controlled objects, while that of *freerefs* is the set of referenced objects that are free to be referenced by other objects. Then in any system the relations

$$ec, \mathit{free} : \mathbb{O} \leftrightarrow \mathbb{O}$$

[1] Attributes of a class are partitioned into primary and secondary. The values of the primary attributes determine the state of an object; the values of the secondary attributes depend upon the primary attributes of this or other objects in the system and enable an object to retain information about various aspects of the state of the system. For a detailed discussion on secondary attributes see [Duke and Rose 1993; Dong, Rose, and Duke 1995].

introduced in Section 3.1 are determined by

$$\begin{array}{l} \forall\, ob_1, ob_2 : \mathbb{O} \bullet \\ \quad ob_1 \; \underline{ec} \; ob_2 \Leftrightarrow ob_2 \in ob_1.\mathit{excon} \\ \forall\, ob_1, ob_2 : \mathbb{O} \bullet \\ \quad ob_1 \; \underline{\mathit{free}} \; ob_2 \Leftrightarrow ob_2 \in ob_1.\mathit{freerefs} \end{array}$$

The properties of the relations *ec* and *free* (as stated in Section 3.1) imply invariant conditions on the system that need not be stated explicitly. In terms of the attributes *excon* and *free* these conditions are:

$$\begin{array}{ll} \forall\, ob : \mathbb{O} \bullet & \texttt{[iia]} \\ \quad ob.\mathit{excon} \cap ob.\mathit{freerefs} = \varnothing & \\ \forall\, ob_1, ob_2 : \mathbb{O} \bullet & \texttt{[iiia]} \\ \quad ob_1 \neq ob_2 \Rightarrow & \\ \qquad ob_1.\mathit{excon} \cap (ob_2.\mathit{excon} \cup ob_2.\mathit{freerefs}) = \varnothing & \end{array}$$

The above predicates can be combined and simplified to give the following predicate:

$$\begin{array}{ll} \forall\, ob_1, ob_2 : \mathbb{O} \bullet & \texttt{[iv]} \\ \quad ob_1.\mathit{excon} \cap ob_2.\mathit{freerefs} = \varnothing & \\ \quad (ob_1 \neq ob_2 \;\Rightarrow & \\ \qquad ob_1.\mathit{excon} \cap ob_2.\mathit{excon} = \varnothing) & \end{array}$$

This predicate is a global invariant of any Object-Z specification.

Considering again the piggy-bank example in Section 2, using this global invariant the exclusive control relation between single people and their piggy-banks can be ensured by specifying the class *SinglePerson* as:

$$\begin{array}{l} \underline{\mathit{SinglePerson}} \\ \quad \begin{array}{|l} \mathit{has}_a : \mathit{PiggyBank} \\ \hline \mathit{excon} = \{\mathit{has}_a\} \\ \mathit{freerefs} = \varnothing \end{array} \\ \quad \ldots \end{array}$$

If the role of an attribute is to always identify exclusive controlled objects, this can be indicated when the attribute is declared by appending a subscript '$_{\textcircled{\scriptsize E}}$' to the appropriate type. This removes the necessity to write explicit predicates involving *excon* and *freerefs*. For example, adopting this syntactic convention, the class *SinglePerson* becomes

$$\begin{array}{|l}
\hline \textit{SinglePerson} \\
\begin{array}{|l}\hline has_a : PiggyBank_{\textcircled{\tiny E}} \\ \hline\end{array} \\
\ldots \\
\hline
\end{array}$$

As object containment notation '$_{\textcircled{\tiny C}}$' does not restrict object access, contained objects are part of the free referenced objects, i.e.

$$\forall\, ob : \mathbb{O} \bullet ob.dcontain \subseteq ob.freerefs$$

where *dcontain* denotes the set of objects directly contained in *ob* (see [Dong and Duke 1995a]).

Type declarations involving '$_{\textcircled{\tiny E}}$' and '$_{\textcircled{\tiny C}}$' can be converted into declarations that directly use the attribute *excon*, *dcontain* and *freerefs* instead; for instance, suppose a class A is defined as:

$$\begin{array}{|l}
\hline A \\
\begin{array}{|l}\hline \ldots \\ \hline\end{array} \\
OpA \mathrel{\widehat{=}} \ldots \\
\hline
\end{array}$$

Then

$$\begin{array}{|l}
\hline B \\
\begin{array}{|l}\hline as_1 : \mathbb{P}\, A \\ as_2 : \mathbb{P}\, A_{\textcircled{\tiny C}} \\ as_3 : \mathbb{P}\, A_{\textcircled{\tiny E}} \\ \hline\end{array} \\
\ldots \\
\hline
\end{array}$$

can be converted to

$$\begin{array}{|l}
\hline B \\
\begin{array}{|l}\hline as_1 : \mathbb{P}\, A \\ as_2 : \mathbb{P}\, A \\ as_3 : \mathbb{P}\, A \\ \hline freerefs = as_1 \cup as_2 \\ dcontain = as_2 \\ excon = as_3 \\ \hline\end{array} \\
\ldots \\
\hline
\end{array}$$

3.3 Preventing Indirect Access

The global class invariant [iv] (above) ensures that exclusively controlled objects cannot be directly referenced by objects other than the owner. However, there is an indirect way whereby an object other than the owner can access an exclusively controlled object. For instance, consider the definition of the class C

$$\begin{array}{|l}
\hline C \\
\begin{array}{|l}\hline b : B \\ \hline\end{array} \\
Op \mathrel{\widehat{=}} [a : b.as_3] \bullet a.OpA \\
\hline
\end{array}$$

where the classes A and B are defined in Section 3.2. In effect, an object c of class C can access the exclusively controlled objects of $c.b$. This access path can be blocked if the client object attributes referencing exclusively controlled objects are invisible to the environment. Therefore a general rule would be that

> in every Object-Z class, attributes referencing exclusively controlled objects are hidden from the environment (by not being placed in the class visibility list[2]).

To enforce the above rule in Object-Z specifications, we introduce a class invisibility-list notation '$\not\upharpoonright(\ldots)$' to complement the class visibility-list. For instance, the conversion for the class B above is automatically enhanced with the class invisibility-list. as:

$$\begin{array}{|l}
\hline B \\
\begin{array}{|l}\hline \ldots \\ as_3 : \mathbb{P}\, A_{\textcircled{\tiny E}} \\ \hline\end{array} \\
\ldots \\
\hline
\end{array}$$

can be converted to

$$\begin{array}{|l}
\hline B \\
\not\upharpoonright(as_3) \\
\begin{array}{|l}\hline \ldots \\ as_3 : \mathbb{P}\, A \\ \hline \ldots \\ excon = as_3 \\ \hline\end{array} \\
\ldots \\
\hline
\end{array}$$

[2]A visibility-list introduces a list of features which are accessible via the dot notation.

The notation '$\not\upharpoonright(as_3)$' denotes that the attribute a_3 is not in the visibility list of the class B. This invisibility-list mechanism facilitates the notion of exclusive control by ensuring that no objects other than the owner can directly access an owned object through the dot notation.

In the next section, the notation for exclusive object control is applied to specify a computer file system. This case study illustrates the distinction between exclusive control and object containment.

4 Case Study: Files and Users

Consider a computer system in which a user can both exclusively own some files, and yet share other files with other users. Any file in the system whether shared or not, has one user as nominal owner. A file can be created or deleted by its nominal owner. Each file of a user can be private or public. If a file is private then no other user can access the file, while if the file is public then all users in the system can access the file. A user can change his or her files from private to public and vice-versa. Private files can also be transferred between users. As an example, an object-reference structure of such a system is illustrated in Figure 1. The following is an Object-Z specification of this system.

Firstly, let the skeletal class

$$
\begin{array}{|l}
\textit{File} \\
\hline
\quad \begin{array}{|l}
\hline
\textit{content} : \textit{Text} \\
\hline
\end{array} \\
\\
\quad \begin{array}{|l}
\textit{Update} \\
\hline
\Delta(\textit{content}) \\
\hline
\end{array} \\
\hline
\end{array}
$$

represent a file, where *Text* is a given type.

A computer user can be modelled as:

$$
\begin{array}{|l}
\textit{User} \\
\hline
\quad \begin{array}{|l}
\hline
\textit{my\_private} : \mathbb{P}\,\textit{File}_{\circledE} \\
\textit{my\_public} : \mathbb{P}\,\textit{File}_{\copyright} \\
\Delta \\
\textit{other\_public} : \mathbb{P}\,\textit{File} \\
\hline
\end{array}
\end{array}
$$

$$
\begin{array}{|l}
\textit{AddToMyPriv} \\
\hline
\Delta(\textit{my\_private}) \\
f? : \textit{File} \\
\hline
f? \notin \textit{my\_private} \\
\textit{my\_private}' = \textit{my\_private} \cup \{f?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{DelFromMyPriv} \\
\hline
\Delta(\textit{my\_private}) \\
f? : \textit{File} \\
\hline
f? \in \textit{my\_private} \\
\textit{my\_private}' = \textit{my\_private} \setminus \{f?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{AddToMyPub} \\
\hline
\Delta(\textit{my\_public}) \\
f? : \textit{File} \\
\hline
f? \notin \textit{my\_public} \\
\textit{my\_public}' = \textit{my\_public} \cup \{f?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{DelFromMyPub} \\
\hline
\Delta(\textit{my\_public}) \\
f? : \textit{File} \\
\hline
f? \in \textit{my\_public} \\
\textit{my\_public}' = \textit{my\_public} \setminus \{f?\} \\
\hline
\end{array}
$$

$$
\begin{array}{ll}
\textit{OpenAccess} \mathrel{\widehat=} & \textit{DelFromMyPriv} \\
& \wedge \\
& \textit{AddToMyPub} \\
\textit{LimitAccess} \mathrel{\widehat=} & \textit{DelFromMyPub} \\
& \wedge \\
& \textit{AddToMyPriv}
\end{array}
$$

$$
\begin{array}{|l}
\textit{Select} \\
\hline
f? : \textit{File} \\
\hline
f? \in (\textit{my\_private} \cup \\
\qquad \textit{my\_public} \cup \textit{other\_public}) \\
\hline
\end{array}
$$

$$\textit{Update} \mathrel{\widehat=} \textit{Select} \bullet f?.\textit{Update}$$

The secondary attribute *other_public* denotes all other public files which are not owned by a user but can be accessed by the user. For any user object x, the value of $x.\textit{other\_public}$ is dependent on the environment of x. The precise meaning of this attribute is defined by the state invariant of the system class below.

The system consists of a set of users.

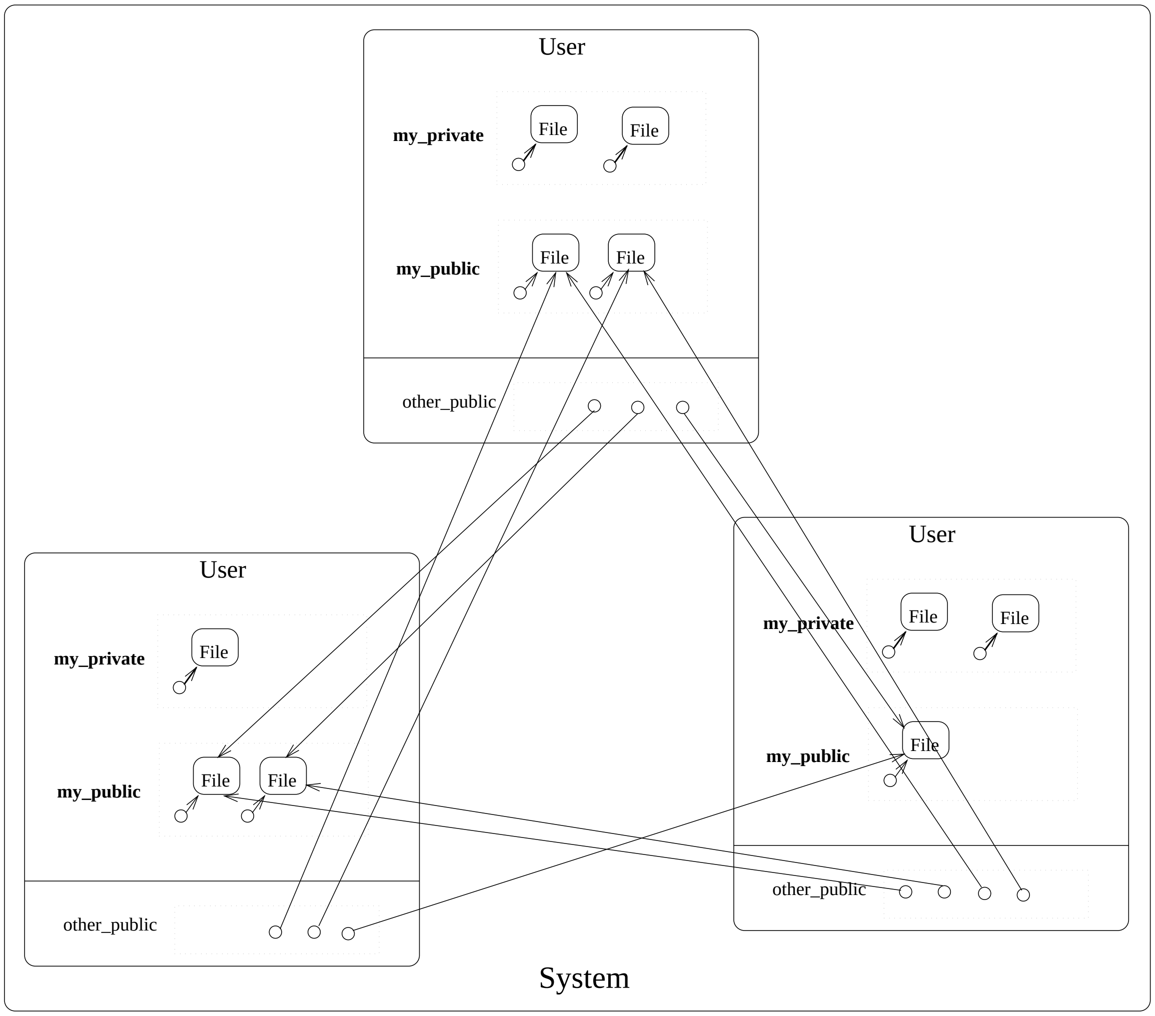

Figure 2: Files and Users.

System

$users : \mathbb{P}\, User$

$$\begin{array}{l} \forall\, u_1 : users \bullet \\ \quad u_1.other\_public = \\ \qquad \bigcup\{u_2 : users \mid \\ \qquad\quad u_2 \neq u_1 \bullet u_2.my\_public\} \end{array}$$

Select

$u? : User$

$u? \in users$

SelectTwo

$u_1?, u_2? : User$

$$\begin{array}{l} u_1? \neq u_2? \\ \{u_1?, u_2?\} \subseteq users \end{array}$$

$$\begin{array}{l} Update \mathrel{\widehat{=}} Select \bullet u?.Update \\ Create \mathrel{\widehat{=}} Select \bullet u?.AddToMyPriv \\ Delete \mathrel{\widehat{=}} Select \bullet u?.DelFromMyPriv \\ OpenAccess \mathrel{\widehat{=}} Select \bullet u?.OpenAccess \\ LimitAccess \mathrel{\widehat{=}} Select \bullet u?.LimitAccess \\ Transfer \mathrel{\widehat{=}} SelectTwo \bullet \\ \qquad\qquad u_1?.DelFromMyPriv \\ \qquad\qquad \| \\ \qquad\qquad u_2?.AddToMyPriv \end{array}$$

The state invariant of *System* implies that if a file is public then all users in the system can access the file.

The operation $Create \mathrel{\widehat{=}} Select \bullet u?.AddToMyPriv$ specifies that a file $f?$ can be added to a user $u?$. Although the specification is not concerned with how the file

$f?$ is created, it precisely captures that the file added to the user is a new file which is not referenced by any user in the system before the operation is invoked. This is because of the explicit precondition $f? \notin u?.my\_private$ and the global class invariant ([iv] in Section 3.2) ensuring that no users other than $u?$ can reference the files of $u?.my\_private$.

The operations *OpenAccess* and *LimitAccess* specify that a file $f?$ of a user $u?$ can be transferred between $u?.my\_private$ and $u?.my\_public$. These two operations illustrate that a client object can change its exclusive control of another object.

The operation *Transfer* specifies that a private file can be transferred from one user to another. This operation illustrates that the unique access right of an object can be transferred from one owner to another.

5 Discussions and Conclusions

Object sharing (aliasing) and object control are both important notions in object-oriented systems, and the formal specification of such systems needs to capture precisely these notions. The reference semantics of Object-Z takes the view that every referenced object is potentially shared. In this paper, we have extended the Object-Z notation to directly capture the notion of exclusive object control. A possible way to implement the exclusive control notion in object-oriented programming languages, such as Eiffel[Meyer 1992], is to use the *expanded* class type.

The notion of exclusive object control is an important aspect of safety and security critical systems. With the specific notation defined in this paper, Object-Z is an ideal language to specify safety and/or security critical systems. A task for future research is to investigate how the notation for exclusive object control can facilitate reasoning about systems specified in Object-Z.

Acknowledgements

We would like to thank Gordon Rose for many helpful discussions on the issues raised in this paper. We would also like to thank Shannen D. Dan for drawing the artistic cartoon for this paper.

References

Atkinson, S. (1995, September). Formalizing the Proposed Eiffel Library Kernel Standard. Technical Report 95-35, Software Verification Research Centre, Dept. of Computer Science, Univ. of Queensland, Australia. To appear in *Proc. Technology of Object-Oriented Languages and Systems: TOOLS 18*, Prentice Hall 1995.

Dong, J. S. Living with Free Type and Class Union. To appear in *The 1995 Asia-Pacific Software Engineering Conference (APSEC'95) Brisbane, December 1995.*

Dong, J. S. (Oct 1995). *Formal Object Modelling Techniques and Denotational Semantics Studies.* Ph. D. thesis, The University of Queensland. Submitted.

Dong, J. S. and R. Duke (1993, November). Class Union and Polymorphism. In C. Mingins, W. Haebich, J. Potter, and B. Meyer (Eds.), *Proc. 12th International Conference on Technology of Object-Oriented Languages and Systems. TOOLS 12 & 9*, pp. 181–190. Prentice-Hall.

Dong, J. S. and R. Duke (1995b). Using Object-Z to Specify Object-Oriented Programming Languages. Technical Report 95-23, Software Verification Research Centre, Dept. of Computer Science, Univ. of Queensland, Australia.

Dong, J. S. and R. Duke (Chapman & Hall, March 1995a). The Geometry of Object Containment. *Object-Oriented Systems 2*(1), 41–63.

Dong, J. S., R. Duke, and G. Rose (1994, January). An Object-Oriented Approach to the Semantics of Programming Languages. In G. Gupta (Ed.), *Proc. 17th Annual Computer Science Conference (ACSC'17)*, NZ, pp. 767–775.

Dong, J. S., G. Rose, and R. Duke (1995, November). The Role of Secondary Attributes in Formal Object Modelling. In A. Stoyenko (Ed.), *The First IEEE Inter-*

national Conference on Engineering Complex Computer Systems (ICECCS'95), Florida, pp. 31–38. IEEE Computer Society Press.

Duke, R., P. King, G. Rose, and G. Smith (1991). The Object-Z specification language. In T. Korson, V. Vaishnavi, and B. Meyer (Eds.), *Technology of Object-Oriented Languages and Systems: TOOLS 5*, pp. 465–483. Prentice-Hall.

Duke, R. and G. Rose (1993, February). Modelling object identity. In *Proc. 16th Australian Comput. Sci. Conf. (ACSC-16)*, pp. 93–100.

Duke, R., G. Rose, and G. Smith. Object-Z: a Specification Language Advocated for the Description of Standards. To appear in a special issue of *Computer Standards and Interfaces* on Formal Methods and Standards, September 1995.

Hussey, A. and D. Carrington (1995). Comparing two user-interface architectures: MVC and PAC. In S. Balbo (Ed.), *QCHI '95 Symposium*, pp. 3–21. Computer Human Interaction Special Interest Group of the Ergonomics Society of Australia.

Meyer, B. (1988). *Object-Oriented Software Construction.* International Series in Computer Science. Prentice-Hall.

Meyer, B. (1992). *Eiffel: The Language.* Prentice-Hall.

Rose, G. (1992). Object-Z. In S. Stepney, R. Barden, and D. Cooper (Eds.), *Object Orientation in Z*, Workshops in Computing, pp. 59–77. Springer-Verlag.

Appendix: Outline of Object Containment

The development of the object containment notation was initially motivated by using Object-Z to define the semantics of programming languages[Dong, Duke, and Rose 1994]. The object containment notation has not only played important roles in the object-oriented definition of programming languages[Dong 1995; Dong and Duke 1995b] but has also proved to be useful for modelling object-oriented systems in general[Atkinson 1995; Hussey and Carrington 1995]. In the following, we outline the object containment notation.

Consider the situation where a country contains a set of states, each state contains a set of cities and each city contains a set of suburbs. A specification in Object-Z would be

Suburb

...

City

$sbs : \mathbb{P}\, Suburb$

...

State

$cts : \mathbb{P}\, City$

$$\forall ct_1, ct_2 : cts \bullet ct_1 \neq ct_2 \Rightarrow ct_1.sbs \cap ct_2.sbs = \varnothing$$

...

Country

$sts : \mathbb{P}\, State$

$$\begin{aligned}&\forall st_1, st_2 : sts \bullet\\ &\quad st_1 \neq st_2 \Rightarrow\\ &\qquad st_1.cts \cap st_2.cts = \varnothing\\ &\qquad \forall ct_1 : st_1.cts;\\ &\qquad\quad ct_2 : st_2.cts \bullet\\ &\qquad\qquad ct_1.sbs \cap ct_2.sbs = \varnothing\end{aligned}$$

...

The class invariant of class *State* specifies that no suburb can be in two distinct cities in a state. Similarly, the class invariant of class *Country* specifies that no city can be in two distinct states of the country.

Clearly, capturing the properties of object containment explicitly in this way is cumbersome, particular if the system is large and complex. Therefore in [Dong and Duke 1995a], the Object-Z notation is extended by appending a subscript '$_{©}$' to the appropriate types, and hence introducing global invariants to remove the necessity to write explicit predicates to capture containment. For example, adopting this syntactic convention, the relevant classes in the above specification become

```
┌─ City ──────────────────────
│ ┌───────────────────────
│ │ sbs : ℙ Suburb©
│ └───────────────────────
└─────────────────────────────
```

$$\mathit{City} : \quad \mathit{sbs} : \mathbb{P}\,\mathit{Suburb}_{\copyright}$$

```
┌─ State ─────────────────────
│ ┌───────────────────────
│ │ cts : ℙ City©
│ └───────────────────────
└─────────────────────────────
```

$$\mathit{State} : \quad \mathit{cts} : \mathbb{P}\,\mathit{City}_{\copyright}$$

```
┌─ Country ───────────────────
│ ┌───────────────────────
│ │ sts : ℙ State©
│ └───────────────────────
└─────────────────────────────
```

$$\mathit{Country} : \quad \mathit{sts} : \mathbb{P}\,\mathit{State}_{\copyright}$$

In general, two properties of object containment are implied when the $_{\copyright}$ abbreviation is used:

(1) no object directly or indirectly contains itself; and

(2) no object is directly contained in two distinct objects.

The semantics behind the syntactic convention '$_{\copyright}$' is that every Object-Z class has an implicitly declared secondary attribute *dcontain* (directly contained objects). Type declarations involving '$_{\copyright}$' can be converted into declarations that directly use the attribute *dcontain* instead; for instance, the state schema of the *Country* class can be converted to

```
┌─────────────────────────────
│ sts : ℙ State
├──────────────
│ dcontain = sts
└─────────────────────────────
```

$$\mathit{sts} : \mathbb{P}\,\mathit{State} \quad | \quad \mathit{dcontain} = \mathit{sts}$$

Properties of object containment are captured by implicit (global) class invariants. For example, the property (2) 'no object is directly contained in two distinct objects' is formalised as the implicit (global) class invariant:

$$\forall\, ob_1, ob_2 : \mathbb{O} \bullet ob_1 \neq ob_2 \Rightarrow ob_1.dcontain \cap ob_2.dcontain = \varnothing$$

Notice that the notions of containment and control are in general quite distinct and should not be confused in system modelling: containment is concerned with the relative geometric, or structural relationship between objects; control is concerned with access, i.e. the right of one object to send a message to another object. Often within object-oriented systems the case arises when one object contained within another can be sent messages by a third object elsewhere in the system. For a detailed treatment of object containment see [Dong and Duke 1995a].