DEPARTMENT OF COMPUTER SCIENCE

Concrete Examples for Templates and Their Children

P.N. Taylor

Technical Report No. 279

April 1997

Concrete Examples for Templates and Their Children

P.N. Taylor

Department of Computer Science, Faculty of Information Sciences,

University of Hertfordshire, College Lane, Hatfield, Herts. AL10 9AB. U.K.

Tel: 01707 284763, Fax: 01707 284303, Email: p.n.taylor@herts.ac.uk

April 4, 1997

Abstract

This paper explains the concepts of template, class, object and type using concrete examples from the theory of sets and natural numbers. The descriptions of these concepts are taken from the Reference Model of ISO's Open Distributed Processing document (RM-ODP) 10746 (Part 2).

The concepts of subtype/supertype and subclass/superclass are also explained in terms of their ODP definitions with simple examples, together with the differences between subtype and subclass.

Introduction

The International Standardization Organization's (ISO) work on the Reference Model for Open Distributed Processing (RM-ODP) is provided to give developers a standard foundation and definition of object-oriented concepts.

In this paper the ODP definitions for template, class, object, type, subtype and subclass are reviewed. We then provide examples of each term. The descriptions for these concepts are taken from the ODP document itself [5] and its interpretation by both Rudkin [10] and Cusack [2].

It is our intention that this paper shall provide the reader with concrete examples of the concepts addressed by ODP in relation to object-oriented design and applied to the notation of CSP-like process algebras [3]. The only prerequisites for an understanding of these concepts, as presented in this paper, are a basic understanding of set and number theory.

Section 1 of this paper introduces the ODP definitions of template, class, object (instance), type, subtype and subclass as they appear in the work of the ODP [5], Rudkin [10] and Cusack [2].

Section 2 presents simple concrete examples of each of the ODP object-oriented definitions listed in section 1 using natural number examples.

Section 3 illustrates what is meant by subtype, supertype, subclass and superclass and distinguishes the differences between these concepts.

Section 4 shows how the ideas of object-oriented specification, as shown in the previous sections, can be applied to process algebra notation to permit the reuse and expansion of existing defined behaviour.

Section 5 presents conclusions and future work from this study with application to the extension of process algebra specifications.

1 ODP Definitions of Object-Oriented Concepts

The following object-oriented definitions are in use by the ODP standard to enforce consistency between developers and specifiers; such is the nature of standardisation. In this paper we concentrate on the following terms together with their ODP definitions (taken from ODP [5], Rudkin [10] and Cusack [2]):

- **Template**: The specification of the common features of a collection of objects. A template is an abstraction of that collection. Templates may be combined according to some calculus. The precise nature of the combination is determined by the specification language used (in this paper logical conjunction is used to expand existing Boolean predicates associated with each template).

- **Class (of objects)**: The set of all objects obtained from a given template (known as the class template) by a process of instantiation. Each object is an instance of the class template.

- **Object (instance)**: An object is an instance of a class when it is related to the class template via some chosen membership relation (i.e: it conforms to the predicate of the class template).

- **Type**: A type is defined to be a predicate which determines the instances of a class. An object is an expression of the type if the predicate of the class template holds for the object.
- **Subtype/Supertype**: If one type implies another then the first type is a subtype of the second type. The second type is a supertype of the first. If every x which satisfies the first type also satisfies the second type then the first type is a subtype of the second type.
- **Subclass/Superclass**: One class is a subclass of another class if the type of the first class is a subtype of the second class. The second class is a superclass of the first class in that case. The relationship between subtypes and subclasses is explained in more detail in section 3.

Each of these concepts can now be illustrated using simple examples from the theory of sets and natural numbers.

2 Concrete Examples of Object-Oriented Concepts

This section presents simple examples of the terms defined in section 1, using elements from the set of natural numbers.

Template

A template T_1 can be defined as a predicate. For example, a simple rule over the set of natural numbers ($\mathbb{N}$) is defined:

$$T_1 \stackrel{\text{def}}{=} n > 5, \text{ where } n : \mathbb{N} \tag{1}$$

Another template T_2 can be defined in a similar way:

$$T_2 \stackrel{\text{def}}{=} n > 10, \text{ where } n : \mathbb{N} \tag{2}$$

Class

With the two templates T_1 and T_2 a class is defined as a set, the elements of which are objects conforming to the rules of their associated template.

$$C_1 = \{6, 7, 8, \ldots\}, \text{ taken from template } T_1$$

$$C_2 = \{11, 12, 13, \ldots\}, \text{ taken from template } T_2$$

Object

Each element of either of the two sets C_1 or C_2 is an object instance. Therefore, a class is defined to be a set of object instances which conform to a class template (i.e: conform to the template's predicate). For example, consider the element 7, drawn from the class C_1. Objects of class C_1 are related to template T_1. The predicate for T_1 is defined as $n > 5$ and 7 meets this criteria, hence 7 is a valid instance of a class described by template T_1.

Type

A type is also defined to be a predicate, hence template and type are related. According to the definition of type 'an object is an expression of the type if the predicate of the class template holds for the object' (as can be seen for 7 being a valid object for template T_1). An object template (like T_1) together with some chosen membership relationship (as defined by the predicate of T_1) is a type.

3 Subtype/Supertype and Subclass/Superclass Relationships

In our simple example T_2 implies T_1 because any object instance that conforms to the predicate $n > 10$ must also conform to the predicate $n > 5$, therefore, by definition, T_2 is a subtype of T_1 (T_1 is by implication a supertype of T_2). We can write the subtype relationship formally as follows:

$$T_2 \Rightarrow T_1, \text{ implies that } T_2 \sqsubseteq_{subtype} T_1$$

If we look at the contents of the sets denoting the instances of objects for the classes related to T_1 and T_2 we can use the subset operator to prove the subtype relationship. Indeed, T_2 is a subset of T_1 ($T_2 \subseteq T_1$). In extension we would write this as follows:

$$\{11, 12, 13, \ldots\} \subseteq \{6, 7, 8, \ldots\}$$

With the use of set notation to describe objects of a certain class we can also use the numerous set theory operators to manipulate the expressions of a class set of object instances, as is the case with the subset relationship defined above over subtypes.

3.1 Incremental Modification and Inheritance

Given that $C_2 \subseteq C_1$ we can further restrict T_1 using incremental modification of the predicate defined in T_1. The reader is referred to the work of Wegner [11] for an in-depth discussion on incremental modification and inheritance. A restriction of T_1 is defined as T_3:

$$T_3 \stackrel{\text{def}}{=} T_1 \wedge Q, \text{ where } Q \stackrel{\text{def}}{=} n > 10 \tag{3}$$

The associated class for T_3, namely C_3, is now represented as $\{11, 12, 13, \ldots\}$, the same as that of C_2. Consequently, $T_3 \equiv T_2$ which allows us to substitute T_3 for T_2 in places where T_2 was originally expected.

Further incremental modification denotes further conjunction between the templates and predicates. For example:

$$T_4 \stackrel{\text{def}}{=} T_3 \wedge R, \text{ where } R \stackrel{\text{def}}{=} n > 12 \text{ and } T_4 \equiv T_3 \equiv T_1 \wedge Q \wedge R \tag{4}$$

Substitution is discussed in more detail in section 3.3 below. This section continues with a focus on the incremental modification of template definitions, namely strict inheritance.

In the first example in this section the template T_3 is defined as an incremental modification of T_1, using logical conjunction with the predicate Q. Within the realm of natural numbers inheritance is implemented using logical conjunction. Incremental modification itself can be referred to as inheritance (or strict inheritance to be more precise). The inheritance hierarchy for the existing templates is shown as follows:

$$T_4 \sqsubseteq_{inherits} T_3 \sqsubseteq_{inherits} T_2 \sqsubseteq_{inherits} T_1$$

Having established the inheritance hierarchy the subtype relationship between the three templates can now be written:

$$T_4 \sqsubseteq_{subtype} T_3 \sqsubseteq_{subtype} T_2 \sqsubseteq_{subtype} T_1$$

Again, using the subset operator a concrete example of the subtype relationship can also be expressed:

$$C_4 \subseteq C_3 \subseteq C_2 \subseteq C_1$$

The previous expression is written in extension as:

$$\{13, 14, 15, \ldots\} \subseteq \{11, 12, 13, \ldots\} \subseteq \{11, 12, 13, \ldots\} \subseteq \{6, 7, 8, \ldots\}$$

As each incremental modification is added to the predicate of the template (by logical conjunction) the object instances in the class set become more restricted. The cardinality of the class sets are therefore reduced. Increased specialisation is not guaranteed as a new predicate in conjunction with an existing predicate may have no effect. For example, given that $T_1 \stackrel{\text{def}}{=} n > 5$ and $T_3 \stackrel{\text{def}}{=} T_1 \wedge n > 10$ then the addition of $T_3' \stackrel{\text{def}}{=} T_3 \wedge n > 7$ will not change the contents of C_3 which remains unchanged as $\{11, 12, 13, \ldots\}$.

So, as you can see, it is not guaranteed that incremental modification will reduce the scope of object instances in a template's class set. What is guaranteed is that incremental modification will not increase the scope of the number of objects captured by a class template.

3.2 Contradiction within Predicates

A contradiction in the predicates of a class template will serve to discount any object instances from the class set. The empty set serves as the bottom of ordered object instances and represents the set of objects that meet the criteria of the template's conjunct predicate; namely no objects whatsoever.

For illustrative purposes consider the following example of a contradictory template definition:

$$T_5 \stackrel{\text{def}}{=} n > 10 \wedge n < 10, \text{ where } n : \mathbb{N} \tag{5}$$

The class template for which is shown as $C_5 = \{\}$ which denotes that there is no value in the set of natural numbers that meets the proposition defined by the predicate of T_5.

3.3 Substitution

In section 3.1 the concept of substitution was introduced. This section expands on that introduction. Template T_3 was capable of replacing template T_2 due to their class sets being equal. What would be the situation if T_1 was used to replace T_2?

Certainly, T_1 contains all of the elements of T_2 as $C_1 \supseteq C_2$. Using our natural number example it is easy to see that all instances of C_2 are also instances of C_1.

Put simply, C_2 is more restrictive than C_1 which means that C_2 is *contravariant* in relation to C_1 (i.e: C_2 cannot provide all of the facilities of C_1).

Contravariance between C_2 and C_1 is acceptable to us but is does highlight the fact that C_1 cannot be used to substitute C_2 in its original state; C_3 is required for the substitution of C_2 as it will only offer less-than-or-equal functionality over C_2.

3.3.1 Contravariance and Covariance

As an aside, let us consider the terms *contravariance* and *covariance* and find simple examples from the domain of natural numbers to illustrate them.

Definition 1 Contravariance: *Arguments of the subtype must be less-than-or-equal to arguments of the supertype.*

Consider two types defined as functions which return the square of any input value; $fa(n)$ and $fb(n)$. The functions are defined as begin capable of receiving arguments in the following ranges:

$$fa(n : \mathbb{N} \mid n > 5)$$
$$fb(n : \mathbb{N} \mid n > 10)$$

We can define a set of possible returned values from each function given the full range of input values.

$$\begin{aligned} fa(6, 7, 8, \ldots) &\longrightarrow \{36, 49, 64, \ldots\} \\ fb\underbrace{(11, 12, 13, \ldots)}_{Contravariance} &\longrightarrow \{121, 144, 169, \ldots\} \end{aligned}$$

If fb is said to be a subtype of fa then the input arguments of fb are *contravariant* because the range of fb is only $121, 144, 169, \ldots$ and does not allow for the full range of inputs found in the supertype fa.

Definition 2 Covariance: *Results of the application of the subtype must be less-than-or-equal to the results obtained from the supertype, given the same arguments.*

Consider again the example of inputs to outputs for each defined type in view of covariance:

$$\begin{aligned} fa(6, 7, 8, \ldots) &\longrightarrow \{36, 49, 64, \ldots\} \\ fb(11, 12, 13, \ldots) &\longrightarrow \underbrace{\{121, 144, 169, \ldots\}}_{Covariance} \end{aligned}$$

The output range of the subtype fb is covariant in relation of fa and is therefore safe. Invalid subtypes result from the contravariant arguments in function subtypes (i.e: more restrictive arguments).

In our simple example, if variables of types fa and fb are defined and type assignment is attempted then subtyping will fail due to contravariance. Consider the final example in this

section:

$$\begin{aligned} a &: fa \;//\text{ variable declaration} \\ b &: fb \;//\text{ variable declaration} \\ &\vdots \\ a &:= b \;//\text{ type assignment} \end{aligned}$$

The application of $a(6)$ will fail as a is assigned b of type fb, where the valid input range of function fb is only $\{11, 12, 13, \ldots\}$. Whereas $a(6)$ is valid before the assignment $(a := b)$ it fails immediately after the assignment. Contravariance is responsible for this failure as it has restricted a to an input range of $\{11, 12, 13, \ldots\}$.

3.4 Defining Subtype and Subclass Relationships

The subtype relationship between T_1 and T_2 has been defined. We have shown that $T_2 \sqsubseteq_{subtype} T_1$ and consequently that $C_2 \subseteq C_1$. This section resolves the issue of subtype and subclass relationships between T_1, T_2 and T_3.

It is not always the case that objects related by a subtype relationship are necessarily also related by a subclass relationship. Consider the diagram in figure 1 which illustrates a partial network of the set of natural numbers. The bold set of numbers in figure 1 represents the famous

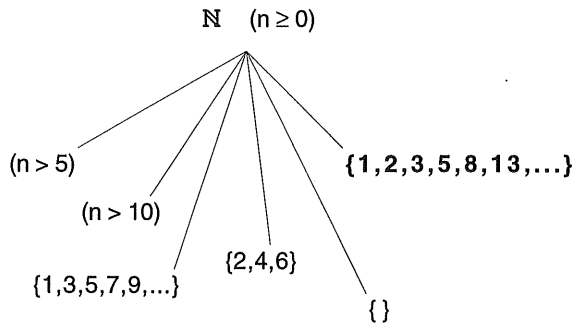

Figure 1: The Set of Natural Numbers

Fibonacci sequence. We have chosen this particular set of values because it represents a valid subtype of natural numbers but not a subclass of the templates T_1 and T_2 as defined by their respective predicates.

Each set present in figure 1 is type compatible as they are all drawn from the set of natural numbers. According to our earlier examples these sets can also be subtype compatible provided

that one is a subset of the other. However, subclass relationships between the sets only apply to all separate sets that meet the requirements of the predicates T_1 and T_2. Hence the exclusion of the set of Fibonacci numbers from the subclass of either T_1 or T_2. The set of odd and even numbers are also excluded from the subclass relationship in this example.

Note: The empty set appears in figure 1 as the set of natural numbers for which no predicate holds. Consider the empty set to be the bottom ($\perp$) of the set of natural numbers. The set governed by the predicate ($n \geqslant 0$) represents the entire set of natural numbers (i.e: the top of $\mathbb{N}, \{0, \ldots, \}$).

Successive incremental modifications of a template, as with T_3, will still not capture some of the sets in the natural number tree represented in figure 1. Extra predicates will enforce more restriction upon the proposition of the template. Logical conjunction is considered to be the basis of incremental modification, with each new additional predicate further specialising the scope of the set of class object instances.

4 Application to Process Algebra Notation

This paper has so far provided a concrete foundation of the principles behind ODP's definitions of template, class, object, type, subtype and subclass. This section relates these concepts to our research work which concentrates upon process algebras and object-oriented specification.

We aim to improve the object-oriented modelling capabilities of CSP-based process algebras by extending a CSP-like language to allow the concepts of template, class and object to be captured. We also intend to address the issue of communications between processes (treated as objects) which must become resilient should synchronisation between such processes fail.

Process algebras such as CCS [9], CSP [3] and LOTOS [4] all pre-date object-oriented design concepts. Naturally, there is no provision for these concepts in such formal languages.

Work is being carried out to address such issues as inheritance [10] to enable formal specifications to benefit from encapsulation and reuse, as object-oriented designed systems currently benefit from such techniques.

Our particular area of interest is inheritance and reuse and the effect that these two areas have over process communication. As we have seen in previous sections, a process (which is treated as an object) has a type. A process definition is specified as a template and therefore a collection of objects conforming to a class template is in fact a collection of processes in a process algebra.

Conformance states that one process must conform to the other if the first process is to replace the second process. If the first (replacement) process fails after some sequence of actions then the original process must also have failed at that point. A formalisation of conformance can now be given.

Definition 3 Conformance

Let Q and P be processes.

(Q conf P) iff

$\forall s \in Traces(P). \forall A \subseteq L(P)$	If Q fails to offer an action a (after sequence
if $\exists Q' \bullet \forall a \in A \bullet Q \xrightarrow{s} Q' \not\xrightarrow{a}$	s) then P must also fail to offer the same
then $\exists P' \bullet \forall a \in A \bullet P \xrightarrow{s} P' \not\xrightarrow{a}$	action a (after the same sequence s).

If the conformance rule is satisfied then a new process can replace an original process in all places where the original process was expected [1]. The environment of the original process will be unaware of the exchange as each function offered by the old process will also be offered by the new process. Conformance guarantees *at least* the same behaviour as was originally present in the environment before the substitution took place.

In CSP notation, conformance between the behaviour of process definitions can be defined via the following rules [1, p.135]:

Law 1 : $\alpha P \subseteq \alpha Q$.

Law 2 : if (s, X) is a failure of Q with $s \in traces(P)$, then (s, X) is a failure of P.

A tuple (s, X) is made up from a trace s of P and a *refusal set* of $P\,(X)$ (after s). A trace is a sequence of (observable) actions for a process.

Law 3 : $(divergences(Q) \cap traces(P)) \subseteq divergences(P)$.

The sequence $divergences(P)$ are the traces of P after which P behaves like CHAOS [2] and can do anything or refuse to do anything; being the most non-deterministic of all processes.

4.0.1 CSP Example of Conformance

A CSP example of conformance and its proof is now given [1, p.135].

$$\alpha P \;=\; \alpha Q = \alpha R = \{a, b\}$$

[1] substitution is discussed further in section 3.3.

[2] $CHAOS_A = \mu X : A.X$

$$
\begin{aligned}
P &= (a \rightarrow STOP) \sqcap (b \rightarrow (a \rightarrow STOP) \,\Box\, (b \rightarrow STOP)) \\
Q &= a \rightarrow STOP \\
R &= (a \rightarrow STOP) \,\Box\, (b \rightarrow (a \rightarrow STOP) \sqcap (b \rightarrow STOP))
\end{aligned}
$$

Process Q conforms (reduces) to P and R conforms (extends) to Q. The failure of $R\,(\langle b \rangle, \{b\})$ is not a failure of P because $\langle b \rangle$ is a trace of both R and P, which confirms that R does not conform to P [1, p.135].

Given that conformance must be present for substitution to result in a stable system we can see the importance of establishing a valid inheritance, subtype and subclass hierarchy, as was shown in the earlier sections of this paper.

4.1 Process Synchronisation

In a process algebra such as CSP processes communicate upon synchronisation between common channels that they both share. When each process is in a position to communicate they will do so together across the shared channel.

If one process is not ready to communicate then the other party will wait until such time as both parties are ready. If one process has died then the other party will wait indefinitely. The entire system may become unstable and deadlock because one component has failed. Clearly, the power to disrupt an entire system should not always be given to each process. However, in an environment of synchronous communications the power to disrupt the execution an entire system is exactly what happens. The nature of the system will determine which processes are essential for the survival of the system and which processes are merely service providers of low priority (such as a print spooler).

We seek to change the method of process communication by introducing a variant of asynchronous communicating processes, using concepts first discussed in the work of Jifeng, Hoare and Josephs [6, 8, 7].

The details of this work on asynchronous communications between processes is beyond the scope of this paper but the concepts of template, class, object and type have far reaching consequences for our own work, hence this discussion. Our main observation is that in order for a process to be modified inheritance must play a part in that modification. Therefore, the mechanisms surrounding inheritance (i.e: incremental modification) must be fully understood.

5 Conclusions and Future Work

This paper has presented concrete examples of object-oriented concepts as defined by ODP's reference model (RM-ODP) and supplemented by the work of both Rudkin [10] and Cusack [2].

Simple examples using sets and natural numbers have been used throughout this paper to illustrate these ODP concepts with a view to providing an easier introduction to the material. The goal of this work has been to provide a firm foundation from which to build up a model of process algebra reuse and the modelling of inheritance.

Our research work uses the concept of inheritance and the asynchronous communications between processes to aid reuse in communicating systems specifications whilst increasing the survivability of those systems should components fail to synchronise. Synchronous communication within a system can lead to *domino waiting* as one process after another fails to provide synchronisation for further processes in the system. Eventually the system becomes wholly unstable and breaks down (i.e: deadlock).

Classifying the concept of a template and then subtypes between class instances, followed by the further classification of subclasses between templates helps in our attempts to successfully model a system that contains inheritance and maintains stability in spite of that inheritance. Hence the importance of this work in ensuring that future work carried out in this area of object-oriented process algebra research starts from a simple, concrete firm foundation which then leads to a flexible unified model.

Acknowledgements

We are am grateful to David Smith for his supervision and insight into the concepts of template, class and type. The example model using natural numbers to illustrate the points made in this paper are based on his observations and our discussions with him. We are also grateful to Mary Buchanan for various discussions on types and subtypes, together with her explanation of contravariance and covariance using the simple example that appears in this paper.

References

[1] E. Cusack. Refinement, Conformance and Inheritance. *BCS Formal Aspects of Computing*, 3(2):129–141, 1991.

[2] E. Cusack, S. Rudkin, and C. Smith. An object-oriented interpretation of LOTOS. In *The 2nd International Conference on Formal Description Techniques (FORTE89)*, December 1989.

[3] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, London, 1985.

[4] Information Processing System Open Systems Interconnection. LOTOS–a formal description technique based on the temporal ordering of observational behaviour. Technical Report DIS 8807, International Standardization Organisation, 1987.

[5] Transfer ISO/IEC Information Retrieval and Management for OSI. Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model. Technical Report ISO/IEC DIS 10746-2, International Standardization Organisation, June 1991. Section 9: Specification Concepts.

[6] H. Jifeng, M.B. Josephs, and C.A.R. Hoare. A theory of synchrony and asynchrony. In M. Broy and C.B. Jones, editors, *Programming Concepts and Methods*, IFIP, pages 459–478. Elsevier Science Publishers B.V: North Holland, 1990.

[7] M.B. Josephs. Receptive process theory. *Acta Informatica*, 29:17–31, 1992.

[8] M.B. Josephs, C.A.R. Hoare, and H. Jifeng. A theory of asynchronous processes. Technical Report PRG-TR-6-89, Programming Research Group, University of Oxford, Oxford University Computing Laboratory, 11 Keble Road, Oxford. OX1 3QD, 1989.

[9] R. Milner. *Communication and Concurrency.* Prentice-Hall, London, 1989.

[10] S. Rudkin. Inheritance in LOTOS. In K.R. Parker and G.A. Rose, editors, *Formal Description Techniques*, volume VI, pages 409–423. Elsevier Science Publishers B.V: North Holland, 1992.

[11] P. Wegner and S.B. Zdonik. Inheritance as an Incremental Modification Technique or What Like is and Isn't Like. In *Proceedings of ECOOP'88*, pages 55–77, Oslo, Norway, August 1988. ECOOP.