A Formal Software Development Approach
Using Refinement Calculus

WANG Yunfeng (王云峰)\textsuperscript{1,2}, PANG Jun (庞军)\textsuperscript{1},
ZHA Ming (查明)\textsuperscript{1}, YANG Zhaohui (杨朝晖)\textsuperscript{1} and ZHENG Guoliang (郑国梁)\textsuperscript{1}

\textsuperscript{1}State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing 210093, P.R. China
\textsuperscript{2}Meteorology College, PLA University of Science and Technology, Nanjing 211101, P.R. China
E-mail: zhenggl@nju.edu.cn

Received July 26, 1999; revised April 14, 2000.

Abstract The advantage of COOZ (Complete Object-Oriented Z) is to specify large scale software, but it does not support refinement calculus. Thus its application is confined for software development. Including refinement calculus into COOZ overcomes its disadvantage during design and implementation. The separation between the design and implementation for structure and notation is removed as well. Then the software can be developed smoothly in the same frame. The combination of COOZ and refinement calculus can build object-oriented frame, in which the specification in COOZ is refined stepwise to code by calculus. In this paper, the development model is established, which is based on COOZ and refinement calculus. Data refinement is harder to deal with in a refinement tool than ordinary algorithmic refinement, since data refinement usually has to be done on a large program component at once. As to the implementation technology of refinement calculus, the data refinement calculator is constructed and an approach for data refinement which is based on data refinement calculus and program window inference is offered.

Keywords formal development method, refinement calculus, formal specification, object-oriented

1 Introduction

The confidence in a program’s correctness can be obtained by describing its intended task in a formal notation. Such specification can then be used as a basis for a provably correct development of the program. The development can be conducted in small steps, thus allowing the unavoidable complexity of the final program to be introduced in manageable pieces.

The process, called refinement, by which specifications are transformed into programs has been extensively studied. (see [1] for an overview of the current work). In particular [2, 3] have laid down much of the theory and have recognized two forms of refinement. The first is algorithmic refinement, by which a program operates more explicitly, usually introducing an algorithm to replace the statement of desired result. The second is data refinement, where one changes the structure for storing information, usually replacing some abstract structure that is easily understood by some more concrete and efficient structure.

The extension from Dijkstra’s language to the refinement calculus was made by Back\textsuperscript{4}, then redeveloped independently by Morris\textsuperscript{5}, Morgan\textsuperscript{6} and by Back and von Wright\textsuperscript{7}.

Gardiner and Morgan\textsuperscript{8} argued a style of data refinements which are calculated directly without proof obligation. It is advantageous over the methods of Spivey\textsuperscript{9} and Stepenny et al.\textsuperscript{10}, which need obligation to prove data refinement.

The work is supported by the National Natural Science Foundation of China (No.69673006) and the National Ninth Five-Year Project (98-780-01-07-06) of China.
Previous attempts to address formal development fall into two categories. In the first one, a specialized calculus is developed within Z or COOZ (Complete Object-Oriented Z) to allow algorithm refinement. An example of this approach can be found in [1]. The second approach involves a translation stage in which the Z specification is transformed into another notation, which is more amenable to refinement. For example, King [12] gave the rules for translating Z specification into Morgan's refinement calculus. Then the refinement calculus law can be used to develop the specification to code.

Both of above approaches seem somewhat wasteful. On one hand, because the translation stage needs much effort but not contributes directly to development, and on the other hand, because so much work has gone into the development of various flavors of refinement calculus which already exist. In particular, Morgan's [13] calculus is well developed and its law has been codified and collected in such a way, which can be used to develop real programs from abstract specification.

So, for the refinement of object-oriented specification, we hope to develop an approach which integrates the Morgan's refinement calculus seamlessly.

Since Z cannot support implementation of the system directly, how to develop executable programs from Z specification has been a valuable research field. The advantage of COOZ [14] is to specify a large scale software, but it does not support refinement of calculating and it needs proof in refinement. But the proof is very hard for OO specification, especially for the large and complex one. Thus its application is confined and it cannot be taken as a whole method for software development. Including refinement calculus into COOZ overcomes its disadvantage during design and implementation. The separation between design and implementation for and notation is removed as well. Then the software can be developed smoothly in the same frame. There is not correspondent object-oriented construct in the existing refinement calculus. The combination of COOZ and refinement calculus can build an object-oriented frame, in which the specification in COOZ is refined stepwise to code by calculus. In the paper, a development model is argued, which is based on COOZ and refinement calculus. The data refinement and operation refinement are analyzed by example; the two methods of operation refinement for OO formal specification is discussed briefly; the frame transition rule from COOZ to C++ is argued. On the implementation technology of refinement calculus, the data refinement calculator is constructed. Finally we argue an approach to data refinement based on data refinement calculus and program window inference.

2 Basic Concept

In this section, some basic concepts are introduced, which include refinement calculus, window inference, program window inference.

2.1 Refinement Calculus

The refinement calculus is based upon the weakest preconditions of Dijkstra, which views programs as predicate transformers, i.e., functions from postcondition to precondition. The refinement calculus extends the guard command of Dijkstra with specification statement. It is a wide spectrum language and a set of correctness preserving rules for deriving executable program from specification. The emphasis on refinement comes from the observation that it is more effective to develop a program and its correctness proof together, as opposed to attempting to verify a given program retrospectively. It is a calculus because the transformation rules calculate the refined program. It includes a specification statement in addition to the usual executable constructs. This integration of specification and execution in one language is the key to a smooth development process, since it allows a program to be de-