软件开发 结构化 形式化 抽象描述 (10)

计算机科学 1995 Vol.22 №.1

47-50

论结构化方法与形式化方法的结合

李景洲 董继润 李保栋

(山东大学计算机科学系 济南 250100)

TP311.52

摘 要 The advantages and drawbacks of both structured method and formal method together with techniques of integrating the two methods are described and analysed in this paper. Two typical classes of techniques for the integration are presented as examples, they are the transformation based and formal framework based techniques. Several problems about the integrated method are disscused at last.

关键词 Structured method, Formal method, Software Engineering.

1. 引 言

结构化方法是目前软件开发的主要方法之一，特别是在近十几年中得到了广泛的应用，而且又有了许多新的发展。几乎在这同时，另一种软件开发方法也正在兴起，并得到应用和发展，这就是形式化软件开发方法。结构化方法具有简单易学、易交流等特点，但缺乏形式化、不严格；而形式化方法一般具有严格的语法和语义，有利于软件生产自动化，但对使用者知识背景要求较高。因此，能否将两种方法相互结合、取长补短这样一个问题就提了出来。

本文对结构化方法和形式化方法分别作了简单介绍和分析，对两种方法结合的技术及优点进行了分析，分别介绍了两种结合的方法作为范例，最后对两种方法结合中的几个问题进行了讨论。

2. 结构化方法

结构化方法主要利用结构化语言、自然语言和图表等来描述需求规格说明，然后再利用一定的方法和步骤，将其逐步转换、过渡到代码。比如 SA/SD 方法中使用了数据流图(DFD)以及实体联系图等，表示系统的动态和静态特性，同时，利用数据字典，结构化语言、PAD 图、程序流程图、模块结构图等，对相关的数据和功能加以定义和描述。还可以将 DFD 转换为模块结构图，由每个模块进而可以编程。

结构化方法特别有利于对系统及其结构的分析。一方面，软件开发人员利用它可以很方便地定义出用户需求，另一方面，还可以直接将其返回给用户以进行正确性和完整性验证。但是结构化方法对系统成分的可重用性及并发性等，都没有明显的方法支持；另外，结构化方法和技术缺乏形式化，其所用符号的简单性和多样性(图表、正文等)，对所给出的需求规格说明很难利用其自身进行正确性、一致性等方面的严格验证，更无法对其进行推理证明，只能由需求分析人员和用户一道进行人工验证。尽管目前提供了许多工具支持辅助这方面的工作，也只是在少数几个环节上可以应用；同样，描述的不精确性和二义性也是结构化方法的一大弱点。

3. 形式化方法

形式化软件开发方法一般借助于抽象描述语言，用严格的数学概念和符号来进行需求规格说明，然后经过逐步求精及变换，最终得到实现的代码。因此，形式化方法的关键是其抽象描述语言。

著名的抽象描述语言有[1]，VDM 方法中基于指称语义的 Meta IV、基于代数语义的 OBJ 和 Larch、基于集合论的 Z、基于范畴论的 Clear，以及中科院研制的基于时序逻辑的抽象描述语言 XYZ/E 等。

根据对抽象类型说明的方式，形式化方法可分为面向模型的方法和面向性质的方法两种[2]。在面向模型的方法中，对抽象类型的说明是为其构造一个模型，该模型的构成成分是一些具有已知特性的数据抽象，例如域、元组、集合、序列、树和映射等，VDM、Z 和 COLD 等都是面向模型的方法；在面向性质的方法中，通过给出一些抽象类型必须满足的性质来定义抽象类型，这些性质通常是以一组公理的形式给出，此种方法的特点是，其规格说明仅描述软件系统的性质，而不涉及其实现方法。

根据抽象描述语言对软件开发各阶段的支持情况，又可分为宽谱(Wide-spectrum)语言和非宽谱语言两种[3]。所谓宽谱语言是指其描述可以具有多种抽象级别，既适用于需求规格说明，又适用于具体设

计和实现，可用于软件开发的各阶段。象 VDM 及 COLD 等都是宽谱语言；非宽谱语言则只适用于软件开发的某个阶段。

利用形式化方法，一方面可以使需求规格说明更精确，可以从数学上对其进行处理，如进行内部一致性和语法正确性的检查或证明，进而可以在一定程度上进行完整性检查[4]；另一方面，借助于形式化描述语言，在由需求规格说明到系统实现的转换过程中，可以避免概念上的模糊与对用户需求的信息转移，从而最终变换为正确的目标代码。

但是，形式化方法最大的问题是要求使用者具备较好的数学背景，这是一般用户和系统开发人员很难达到的，也是妨碍形式化方法推广使用的关键所在。其次，形式化方法大都假定已给定明确的用户需求，这样，开发者必须借助于别的方法获取用户需求，然后再用形式化方法严格定义。

4. 结构化方法与形式化方法的结合

由上面的讨论可以看出，结构化方法与形式化方法各有优缺点，若将两种方法相互结合、取长补短，将是较为理想的，而且这也是完全可行的。

目前，研究者们在两种方法的结合上已做了许多工作[5]。从所结合的形式化方法上看，主要分为基于模型的规格说明技术和基于代数说明的技术。基于模型的技术是与面向模型的抽象描述语言（如 VDM、Z 等）进行结合，这方面的工作始于八十年代后期，研究的结果比较多，比如在 Rolls Royce 的 Yourdon 和 VDM 的结合，在 Tu Delft 的 SA/SD 和 VDM 的结合，Semmens 与 Allen 等人的 Yourdon 和 Z 的结合等；基于代数说明的技术主要是八十年代中期开始的 T. H. Tse 以及 Docker 和 France 等人的工作。从与形式化方法结合时所采用的技术上看，主要可以分为强调语法的和强调语义的两种。比如 T. H. Tse 的工作强调在不同方法之间进行规格说明翻译的能力，即强调语法结构；而 France 和 Docker 的工作则着重于对 SA 方法所模型化的行为特性进行形式化陈述和分析，因而更强调语义方面。

结构化方法与形式化方法结合的一大优势，就在于可以利用结构化方法在需求分析的前端比较容易地获取完整和一致的需求，而将形式化方法的精确性和严格性应用在细节层上，从而保证软件的质量。在何时（即在软件开发的哪一阶段或步骤上）开始应用形式化方法，以及应用到何种程度这些问题上，研究者们所采用的方法也各不相同。一种是结构化方法和形式化方法并行使用，并利用各自的结果互为验证。如 Rolls Royce 的 Yourdon 和 VDM 的结合就属这种情况；另一种是，先用结构化方法，然后再将其结果转换为形式化语言。比如 Nico Plat 等人的工作[6]就属这种情况；还有一种是试图为结构化需求分析提供一个集成的原型开发和形式化说明框架，以建立一个基于 SA 的形式化说明环境。Docker 和 France 在这方面的工作卓有成效，他们也设计了这样一个环境 SAME（Structured Analysis Modeling Environment），而将 SAME 中的模型转换为形式化描述的工作正在进行中[5]。

为了进一步说明两种方法结合的技术和特点，下面对两种典型的结合技术加以分析，即基于转换的 Nico Plat 等人的方法[6]和基于形式化框架的 Docker 和 France 的方法[5]。

5. 基于转换的结合技术 SA/VDM

5.1 基本思想

该方法的基本思想是，将传统的 SA/SD 方法中的 SD 步用 VDM 代替，称为 SA/VDM。首先应用 SA 获得分层的 DFD，然后以系统化、规范化的方法将 DFD 变换为 VDM，从而为开发的软件系统产生一个形式化说明，由于 VDM 的说明语言 Meta IV 是一种宽谱语言，此后便可连续应用 VDM 方法。

5.2 DFD 到 VDM 转换的策略

SA/VDM 方法中，要求 SA 的结果是分层 DFD，在将 DFD 向 VDM 转换时，是从 DFD 低层的基本加工开始的。转换的方法有两种，一种是并行存取方法 PAC，另一种是串行存取方法 SAC。PAC 方法较适合于为 DFD 提供形式化语义，这是因为它所描述的与大多数系统分析员所赋予 DFD 的直观语义是非常相近的；SAC 方法则更倾向于一种转换 DFD 的操作方法，因而对软件开发较为实用。

PAC 和 SAC 方法中都提供了 DFD 到 VDM 转换的规则。下面只对 PAC 方法中有关 DFD 成分到 VDM 构件之间的映射关系加以描述。

5.3 DFD 到 VDM 转换的 PAC 方法

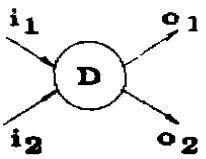
（1）数据存贮。PAC 方法中将数据存贮映射到 VDM 的 State 构件：

```
State NAME of
        ds : DS
end
```

（2）加工与外部实体。在 PAC 方法中，加工和外部实体同等对待，因而此处只讨论加工。加工被映射到 VDM 的 Operation 构件，且使用 VDM 的隐式操作（implicit operation），主要是为了强调加工做什

么，而不是怎样完成的。加工的输入输出分别映射到VDM操作的输入参数和输出参数，这些参数的类型信息可以从用VDM方法定义好的数据词典中获得。

例1　下面是一个DFD片段及其对应的VDM描述：



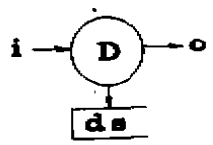
D(i₁ : I₁, i₂ : I₂) o : O₁ × O₂
post let mk-(o₁, o₂) = o in
$P_D(i_1, i_2, o_1, o_2)$

此处P_D是一个谓词，它依赖于加工D的功能特性，不能从DFD中获得，只能由分析设计人员提供。

(3)数据流。数据流表示了加工之间或加工与外部实体之间的连接，并不关心数据是何时或如何流动的，这种连接表示了加工的一种合成，由于两个加工的合成又是一个高层加工，且加工是映射到VDM的操作，所以也将数据流映射到VDM的操作，合成加工所映射的前置条件中应表示出这种连接的存在。

(4)加工与数据存贮。加工与数据存贮的结合被描述为存取系统状态的操作。

例2　加工与数据存贮的VDM描述：



D(i : I) o : O
ext wr ds : DS
post $P_D(i, o, ds, \overleftarrow{ds})$

此处$\overleftarrow{ds}$表示加工存取ds之前的状态。

5.4　特点

这一方法与原SA过程相比具有如下优点：

·为分析设计人员构造系统的形式化说明提供了一个结构化的指导，然后，形式化说明便可用作系统进一步开发的起点。

·为分析设计人员提供了对系统的两种观察，即图形(DFD)和文本(形式化说明)，这可以使他们集中于所关心的方面。

·形式化说明可以被认为是DFD的形式化语义，从而使DFD的含义变得精确，可以对其进行严格的一致性检查。

但是，由于该方法要求用VDM对所有数据类型进行定义，而且操作的谓词描述也只能由分析设计人员给出，因而对软件开发人员的知识背景要求还是较高。

6. 基于形式化框架的结合技术

6.1　基本思想

该方法的基本思想是通过为SA工具提供一个集成的原型开发和形式化说明框架，来建立一个基于SA的形式化说明环境。

该方法提供了两种框架，一种是支持顺序系统的形式化说明，另一种是支持非顺序系统的形式化说明。两种框架都对应用特性的形式化分析与研究有很大帮助，并为形式化分解与求精和形式化验证奠定了良好的基础。

在顺序系统中，DFD被看作是这样一个系统，即一些原子操作存取共享数据结构的实例，共享数据结构用数据存贮描述，而原子操作用加工描述。在非顺序系统中，DFD中加入了控制方面的信息。

下面只对支持顺序系统的框架加以描述。

6.2　顺序系统的语义说明

该框架中描述DFD语义的说明称为原子层说明(Aspec)。DFD的Aspec由两部分组成，数据域定义部分和操作说明(Ospec)部分。数据域定义部分又由数据定义、全局状态定义和全局状态限制组成。

下面结合一下DFD实例来说明各成分的描述。

例3　图书馆借阅图书系统。其DFD如下：

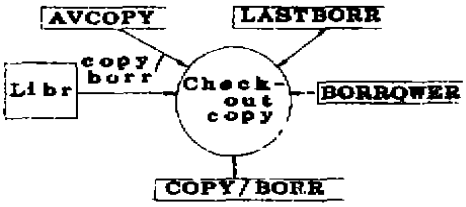


图1

(1)数据定义。该部分包含DFD中所有数据元素的类型定义，每一种类型定义实际上是一种分层的代数说明，它是以创建和使用该类型实例的操作进行类型定义的，加工的输入输出效果只能由该类型定义中所定义的操作来表达。

(2)全局状态定义。该部分由与DFD中数据存贮相关联的数据结构的定义组成，与数据存贮相关联的数据结构的实例称为该数据存贮的状态。状态定义是代数说明的表示，代数说明中是用产生和使用其实例的操作来定义状态的。加工对状态的修改也只能用这些说明中所定义的操作来表达。

如图1中的数据存贮COPY/BORR，它由二元组copy/borr＝〈copy，borrower〉组成，其中，copy是图书馆藏书的类型，borrower是借阅人的类型：

COPY/BORR : set(copy/borr)
Access Operations

```
Signature
  get : set(copy/borr)borrower→set(copy)
  ...
Axioms
For all c : copy/borr; S : set(copy/borr); b : borrower
  1. get(θ,b)=θ
  2. c.borrower=b⇒get(insert(c,S),b)=
     insert(c.copy,get(S,b))
  3. c.borrower≠b⇒get(insert(c,S),b)=get(S,b)
  ...
```

(3)全局状态限制。该部分给出了数据存贮的状态之间必须满足的一些关系，这些限制是用一阶谓词逻辑描述的。

(4)操作说明。Aspec 的操作说明部分由称为 Ospec 的说明组成，每个 Ospec 唯一地与一个加工所描述的操作相关联，Ospec 刻画了一个操作应具有的前置和后置条件，这与 SA 中的加工说明类似。Ospec 中给出的前置后置条件决定了一个加工对一个 DFD 全局状态的影响。

例如图 1 中 Checkout copy 操作的 Ospec 如下：

```
Ospec Checkoutcopy
Input
  c : copy/borr; AVCOPY_in : set(copy);
  BORROWER_in : set(borrower);
  LASTBORR_in : set(copy/borr); COPY/BORR_in
                : set(copy/borr)
Output
  AVCOPY_out : set(copy); LASTBORR_out : set
               (copy/borr);
  COPY/BORR_out : set(copy/borr)
Operation I/O definition
  c.copy∈AVCOPY_in, c.borrower∈BORROWER_in,
  count(get(COPY/BORR_in, c.borrower))<max
⇒COPY/BORR_out=insert(c,COPY/BORR_in),
  AVCOPY_out=delet(AVCOPY_in, c.copy),
  LASTBORR_out=updateId(LASTBORR_in, c)
```

前置条件：要借阅的书(c.copy)必须能够借阅，借阅者(c.borrower)必须是已注册的，而且当前借出的书数量必须严格小于 max。

后置条件：输入 c 被存贮在 COPY/BORR 中，借出的书(c.copy)成为不可再借的，该书的"last borrower"关系被更新为新的借阅者。

6.3 特点

该方法将 SA 与形式化说明融为一体，使得在应用 SA 的同时获得形式化说明，是一种较为理想的软件开发范式。

Docker 和 France 设计的 SAME 为该方法的具体应用提供了良好的支撑环境。SAME 中提供了 DFD 作图、数据定义，作为原型的应用模型开发并允许不完整模型的运行。特别是 SAME 的"soft-fail"运行环境，提供了错误陷井，允许用户对错误进行分析并纠错，然后再从错误点处继续执行。这对应用快速原型技术提供了较好的支持。

7. 问题讨论

由前面的论述可以看出，结构化方法与形式化方法的结合，确实克服了两种方法各自的许多缺点，并起到了取长补短的作用。但也还存在着一些问题，下面仅就少数几个问题加以讨论。

(1)分析阶段的开销将会更大。由于在应用结构化方法时要和形式化方法接轨，无论对功能还是数据方面的要求，都要以形式化描述为基础，这就比传统的 SA 方法更为严格、细致，这必然导致时间和精力方面支出的增加。不过，一旦获得较为准确的用户需求后，其后的开发过程将是自动化程度较高的，因而这种开销也是合算的。

(2)需要大量的工具支持。所以要将结构化方法与形式化方法结合，就是因为一旦获取形式化规格说明后，可以经过变换自动产生代码，这当然需要大量分析、变换工具的支持。另一方面，在将结构化方法结果向形式化说明过渡，以及所有结果文档的存贮管理、一致性检查和原型开发等，都离不开工具，象 SAME 这样的环境是必须的。

(3)缺乏对系统实施细节的表述能力。象上面例举的两种方法中，能够表达设计方面内容的，主要是允许在 DFD 中加入控制信息，象系统性能及系统运行环境、资源等方面的要求，都很难表达。不过这也是结构化方法和形式化方法自身所具有的缺陷。

虽然存在一些问题，但结构化方法与形式化方法结合所具有的优点是显而易见的，两种方法的结合应用，必将对提高软件生产率和软件质量，进而对迈向软件生产自动化产生巨大的推动作用！

参考文献

[1]唐稚松，程序技术研究三十年，计算机科学，1988,3

[2]R. J. Bril, A Model-oriented Method for Algebraic Specifications Using COLD-1 as Notation, LNCS 551, Springer-Verlag, 1991

[3]陈万军，张然，论形式化的软件规格说明方法，计算机应用与软件，1991,5

[4]M. D. Fraser et al., Informal and Formal Requirements Specification Language, Bridging the Gap, IEEE Transactions on Software Engineering, 1991, Vol. 17, No. 5

[5]L. T. Semmens, Integraed Structured Analysis and Formal Specification Techniques, The Computer Journal, 1992, Vol. 35, No. 6

[6]Nico Plat, A Case for Structured Analysis/Formal Design, LNCS 551, Springer-Verlag, 1991