Operational specification for FCA using Z

ANDREWS, S. <http://orcid.org/0000-0003-2094-7456> and POLOVINA, S. <http://orcid.org/0000-0003-2961-6207>

Operational Specification for FCA using Z

Simon Andrews and Simon Polovina

Faculty of Arts, Computing, Engineering and Sciences
Sheffield Hallam University, Sheffield, UK
{s.andrews, s.polovina}@shu.ac.uk

Abstract. We present an outline of a process by which operational software requirements specifications can be written for Formal Concept Analysis (FCA). The Z notation is used to specify the FCA model and the formal operations on it. We posit a novel approach whereby key features of Z and FCA can be integrated and put to work in contemporary software development, thus promoting operational specification as a useful application of conceptual structures.

1 Introduction

The Z notation is a method of formally specifying software systems [1, 2]. It is a mature method with tool support [3] and an ISO standard[1]. Its strength is in providing a rigorous approach to software development. Formal methods of software engineering allow system requirements to be unambiguously specified. The mathematical specifications produced can be formally verified and tools exist to aid with proof and type checking. Being based on typed set theory and first order predicate logic, Z is in a position to be exploited as a method of specification of systems modeled using FCA.

An issue with formal methods has been the amount of effort required to produce a mathematical specification of the software system being developed. Having a 'ready made' mathematical model provided by FCA would allow formal methods to have a new outlet. Whilst FCA can already be used to aid in the understanding and implementation of software systems (see next Section), Z can provide the method and structure by which FCA can be properly integrated into a development life cycle.

Work linking FCA and Z has been undertaken [4] that uses FCA as a means by which Z specifications can be explored and visualised. However, it does not appear that the link has been established in the other direction, i.e. that an FCA model can be taken as a starting point for functional requirements specification in Z. We are interested in specifying functional system requirements as operations on the FCA data model, thus allowing the strengths of FCA and Z to be combined. Work on algorithms based on FCA has been carried out, for example by Carpineto and Romano [5], but here we are suggesting a formal approach to

[1] Information Technology - Z Formal Specification Notation - Syntax, Type System and Semantics, ISO/IEC 13568:2002

the abstract specification of system requirements that can assist in transforming the conceptual model into an implementation.

2 FCA in Software Development

FCA has been used in a number of ways for software development; for modeling the data structure of software applications, such as ICE [6], DVDSleuth [7] and HierMail [8], and as the basis for specialised application building environments such as ToscanJ [9] and Galicia [10]. However there appears to be little work concerning the use of FCA as part of a general software engineering life cycle.

Tilley *et al* [11] have conducted a survey of FCA support for software engineering activities which found that the majority of reported work was concerned with object-oriented re-engineering of existing/legacy systems and class identification tasks. They found little that related to a wider software engineering context or to particular life cycle phases.

One piece of work that does relate FCA directly to phases of the software life cycle has been carried out by Hesse and Tilley [12]; They discuss how FCA applies to requirements engineering and analysis. By taking a use-case approach, relating information objects to functional processes, they show that a hierarchical program structure can be produced. They suggest that FCA can play a central role in the software engineering process as a form of concept-based software development. The approach of this paper embodies their idea, with FCA providing the information structure and Z providing the process specification (Figure 1).

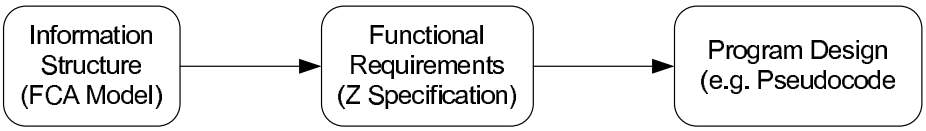

Fig. 1. FCA and Z in the Software Life Cycle

3 From FCA to Z

In FCA a *formal context* consists of a set of objects, G, a set of attributes, M, and a relation between G and M, $I \subseteq G \times M$. A *formal concept* is a pair (A, B) where $A \subseteq G$ and $B \subseteq M$. Every object in A has every attribute in B. For every object in G that is not in A, there is an attribute in B that that object does not have. For every attribute in M that is not in B there is an object in A that does not have that attribute. A is called the *extent* of the concept and B is called the *intent* of the concept. If $g \in A$ and $m \in B$ then $(g, m) \in I$, or gIm.

In Z, information structures are declared based upon a typed set theory. To apply this in FCA, G becomes such a type, namely the universal set of objects of interest. Similarly, M becomes the universal set of attributes that the objects of interest may have. The notation $g : G$ declares an object g of type G and $m : M$ declares an attribute m of type M. Sets can be declared using the powerset notation, $\mathbb{P}$, and relations declared by placing an appropriate arrow between related types.

3.1 Formal Context as a System State

Using the Z notation, the formal context and concepts can be specified as *state variables* in a *state schema* (Figure 2), declaring the relation I, along with a *concept function*, S, which maps extents to intents. S is declared as an injection; an intent has one and only one extent, an extent has one and only one intent.The lower section of the schema (the schema predicate) logically describes how I and S are related. $A : \mathbb{P}\, G$ declares that A is a set of objects. B is the intent of A. |

ContextAndConcepts

$$I : G \leftrightarrow M$$
$$S : \mathbb{P}\, G \rightarrowtail \mathbb{P}\, M$$

$$\forall A : \mathbb{P}\, G;\ B : \mathbb{P}\, M \mid (A, B) \in S \bullet$$
$$\forall g : G;\ m : M \bullet g \in A \land m \in B \Leftrightarrow gIm \land$$
$$\forall g : G \mid g \notin A \bullet \exists m : M \mid m \in B \land \neg\, gIm \land$$
$$\forall m : M \mid m \notin B \bullet \exists g : G \mid g \in A \land \neg\, gIm$$

Fig. 2. State Schema specifying a Formal Context and its Concepts

can be read as 'such that' and $\bullet$ can be read as 'then'.

Although a proof is not attempted here, the predicate appears, by inspection, to satisfy Wille's conditions for deriving concepts so that $A = B^I$ and $B = A^I$ [13].

3.2 Query Operations

In Z, a query postfix, ?, is used to indicate an input to an operation and an exclamation postfix, !, is used to indicate an output from an operation. The symbol Ξ indicates that the operation does not change the value of the state variables.

In Z, if R is a binary relation between X and Y, then the *domain* of R ($\mathrm{dom}\, R$) is the set of all members of X which are related to at least one member of Y by R. The *range* of R ($\mathrm{ran}\, R$) is the set of all members of Y to which at least one member of X is related by R.

By making use of the concept function, S, and the fact that it is injective, operations to output the intent of an extent and to output the extent of an intent, are easily specified. Figure 3 specifies the latter in an operation schema called *FindExtent*.

A strength of the Z notation is its notion of preconditions and postconditions. Preconditions are statements that must be true for the operation to be successful and postconditions specify the result of the operation. In *FindExtent*, the precondition $B? \in \operatorname{ran} S$ states that the input set of attributes must be in the range of S. The postcondition $A! = S^{\sim}(B?)$ obtains the extent by inverting S and supplying it with the intent.

FindIntent is not specified here as it is, essentially, a mirror of *FindExtent*, with the input being a set of objects and the output being the corresponding set of attributes, $B! = S(A?)$.

FindExtent

$\Xi ContextAndConcepts$
$B? : \mathbb{P}\, M$
$A! : \mathbb{P}\, G$

$B? \in \operatorname{ran} S$
$A! = S^{\sim}(B?)$

Fig. 3. An operation to find the extent of an intent

A query operation that outputs an object's attributes, called *FindAttributes* is shown in Figure 4. The set of attributes is obtained by taking the relational image of I through a set containing the object of interest. Again, the operation *FindObjects* (for an attribute of interest) is similar and is not specified here.

FindAttributes

$\Xi ContextAndConcepts$
$g? : G$
$B! : \mathbb{P}\, M$

$g? \in \operatorname{dom} I$
$B! = I(\!|\, \{g?\} \,|\!)$

Fig. 4. An operation to find an object's attributes

Operation schemas to find object concepts and attribute concepts can be specified according to Wille's definitions, $\gamma g := (\{g\}^{II}, \{g\}^{I})$ and $\gamma m := (\{m\}^{II}, \{m\}^{I})$,

by piping together the corresponding object/attribute, extent/intent queries using a chevron notation, >>. The output from the schema preceding the chevrons becomes the input for the schema that follows them:

$$FindObjectConcept \mathrel{\widehat=} FindAttributes >> FindExtent,$$
$$FindAttributeConcept \mathrel{\widehat=} FindObjects >> FindIntent.$$

In each case, we are interested in the outputs of both of the piped schemas, so that $\gamma g = (A!, B!?)$ and $\gamma m = (A!?, B!)$. The postfix !? indicates that something is first an output and then an input.

3.3 Update Operations

A strength of the Z notation is its notion of *before* and *after* states, i.e. a clear distinction is made between the value of state variables before an operation is carried out and their values after the operation is carried out. A state variable decorated with an apostrophe indicates that is in the *after* state. The symbol Δ indicates that an operation changes the state.

An operation to add a new object to the context can be specified by declaring the object and the object's attributes as inputs. The operation schema *AddObject* is shown in Figure 5. It is a precondition that the attributes currently exist in the context.

In Z, $\rhd$ subtracts elements from a range and $\triangleright$ restricts a range. These are used in the postcondition involving S to take into account the possibility that the attributes of the new object are an existing intent. The relevant concept is updated by adding the new object to the corresponding extent.

AddObject

$\Delta ContextAndConcepts$
$g? : G$
$B? : \mathbb{P}\, M$

$g? \notin \operatorname{dom} I$
$B? \subseteq \operatorname{ran} I$
$I' = I \cup \{ m : M \mid m \in B! \bullet g? \mapsto m \}$
$S' = (S \rhd \{B?\}) \cup \{\bigcup(\operatorname{dom} S \triangleright \{B?\}) \cup \{g?\} \mapsto B?\}$

Fig. 5. An operation to add a new object

A similar operation to add a new attribute can be specified, but is not given here. Other useful operations that can be specified include those to remove an object from the context, remove an attribute from the context, remove an attribute from an object, remove an object from an attribute and to add an existing attribute to an existing object. It also is possible that other notions in FCA, such

as the superconcept/subconcept relationship and attribute/object implications, will lend themselves to operational specification in Z.

4 A User Profile Example

Consider a user profile system where users belong to groups and groups are associated with services. The contexts for this system are

$$usergroupContext : USER \leftrightarrow GROUP$$
$$groupserviceContext : GROUP \leftrightarrow SERVICE$$

The complete state schema *UserProfileSystem* is not given for the sake of brevity. The concept functions are also omitted (in practice, where concepts are explicitly required, it may be more pragmatic to specify an axiom to obtain them from the context, rather than include them explicitly in the system state).

An operation is required to form a new group from all users who have access to a particular set of services. The preconditions are that the group must not already exist and that there must be at least one user who has access to the set of services (this also ensures that the services exist). The requirement is specified in Figure 6.

FormGroup

$$\Delta UserProfileSystem$$
$$newgroup? : GROUP$$
$$services? : \mathbb{P}\, SERVICE$$

$$newgroup? \notin \mathrm{dom}\, groupserviceContext$$
$$usergroupcontext \fatsemi groupservicecontext \rhd services? = \emptyset$$
$$\exists\, user : USER \mid services? \subseteq$$
$$\quad \mathrm{ran}(\{user\} \lhd usergroupContext \fatsemi groupserviceContext)$$
$$usergroupContext' = usergroupContext \cup \{user : USER \mid services? \subseteq$$
$$\quad \mathrm{ran}(\{user\} \lhd usergroupContext \fatsemi groupserviceContext) \bullet user \mapsto newgroup?\}$$
$$groupserviceContext' = groupserviceContext \cup$$
$$\quad \{service : SERVICE \mid service \in services? \bullet newgroup? \mapsto service\}$$

Fig. 6. An operation to form a new group in the user profile system

Relational composition is carried out using $\fatsemi$, here to form the relation between users and services. $\lhd$ is domain restriction. Set comprehension is used in the postconditions in the form $\{... \bullet x \mapsto y\}$. The mapping $x \mapsto y$ defines the form of the elements of the comprehended set.

The above example shows how formal contexts, arising from FCA, can be used in the formal specification of system requirements. The operation schema *FormGroup* is an unambiguous specification that can be translated into a program design.

5 Conclusion

The work presented here paves a way by which an FCA model can be specified as a Z state schema and that operations on the model (system requirements) can then be specified as Z operation schemas. The strengths of the conceptual model are thus combined with the strengths of structured formal methods. The Z notation is well understood as part of the software life cycle; it has strengths in the way functional system requirements are structured as schemas and in its notions of pre and post conditions and before and after states. Z has an industry standard and is supported by a variety of software engineering tools. We therefore envisage FCA systems that are specified using Z as opening up FCA to the comprehensive tools and support that are available for Z and vice versa, thereby promoting operational requirements specification as a useful application of conceptual structures.

References

1. J. M. Spivey. The Z Notation: A Reference Manual. Prentice-Hall, 1989.
2. J. B. Wordsworth. Software Development with Z: A Practical Approach to Formal Methods in Software Engineering. Addison-Wesley, 1992.
3. http://vl.zuser.org/#tools (Accessed 21 April 2008).
4. T. Tilley. Towards an FCA based tool for visualising formal specifications. In B. Ganter and A. de Moor, editors, *Using Conceptual Structures: Contributions to ICCS 2003*, pages 227-240. Shaker-Verlag, Germay, 2003.
5. C. Carpineto and G. Romano. Concept Data Analysis: Theory and Application. Wiley, 2004.
6. Y. Qian and L. Feijs. Step-wise Concept Lattice Navigation. In B. Ganter and A. de Moor, editors, *Using Conceptual Structures: Contributions to ICCS 2003*, pages 255-267. Shaker-Verlag, Germay, 2003.
7. J. Ducrou. DVDSleuth: A Case Study in Applied Formal Concept Analysis for Navigating Web Catalogs. In U. Priss, S. Polovina and R. Hill, editors, *Conceptual Structures: Knowledge Architectures for Smart Applications*, ICCS 2007 proceedings, pages 496-500, Springer, 2007.
8. R. Cole, P. Eklund and G. Stumme. Document Retrieval for Email Search and Discovery using Formal Concept Analysis. *Applied Artificial Intelligence*, Volume 17, Number 3, 2003.
9. P. Becker and J. Correia. The ToscanaJ Suite for Implementing Conceptual Information Systems. In B. Ganter, G. Stumme and R. Wille, editors, *Formal Concept Analysis: Foundations and Applications*, pages 324-348. Springer-Verlag, Germany, 2005.
10. P. Valtchev, D. Grosser, C. Roume and M. Hacene. Galicia: an open platform for lattices. In B. Ganter and A. de Moor, editors, *Using Conceptual Structures: Contributions to ICCS 2003*, pages 241-254. Shaker-Verlag, Germay, 2003.
11. T. Tilley, R. Cole, P. Becker and P. Eklund. A Survey of Formal Concept Analysis Support for Software Engineering Activities. In B. Ganter, G. Stumme and R. Wille, editors, *Formal Concept Analysis: Foundations and Applications*, pages 250-271. Springer-Verlag, Germany, 2005.

12. W. Hesse and T. Tilley. Formal Concept Analysis Used for Software Analysis and Modelling. In B. Ganter, G. Stumme and R. Wille, editors, *Formal Concept Analysis: Foundations and Applications*, pages 288-303. Springer-Verlag, Germany, 2005.
13. R. Wille. Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies. In B. Ganter, G. Stumme and R. Wille, editors, *Formal Concept Analysis: Foundations and Applications*, pages 1-33. Springer-Verlag, Germany, 2005.