# Applications of VDM in Banknote Processing

Paul R. Smith[1] and Peter Gorm Larsen[2]

[1] GAO, Euckenstrasse 12, D-81369 München, Germany, Smith@gao-munich.com
[2] IFAD, Forskerparken 10A, DK-5230 Odense M, Denmark,peter@ifad.dk

**Abstract.** We report on the industrial application of VDM in an application for banknote processing. The SIC2000 project at the GAO involved the development of a complex collection of mutually suspicious cooperating software components in a banknote processing system. The role of the formal specification in the project was twofold. It provided a framework in which explicit software invariants are used to reduce the potential of errors arising from the interplay of the cooperating components. It also enabled the construction of an executable system model during the process of designing an adaptive sensor integration subsystem. We present the lessons learned from this project and outline the expected perspectives for the future use of this kind of technology.

## 1 Introduction

This paper reports on the use of VDM in the SIC2000 (Sensor Integration Controller) project, which was undertaken at the GAO (Gesellschaft für Organisation und Automation) with the goal of expediting the integration of sensor software and hardware into existing banknote processing systems. This was the first large project at the GAO in which formal methods played a central role. The analysis, design, and test phases of the development were supported by the IFAD VDM-SL Toolbox [3, 5].

The GAO, which is the currency automation division of Giesecke & Devrient, controls 75% of the world market for banknote processing machinery at central banks, and is currently expanding rapidly into the commercial banking and cash handling market. The primary source of GAO's turnover are the banknote processing systems, of which there are a half-dozen families, distinguished from one another by their size, functionality, processing speed, and the technological era in which they were developed [2].

The application we describe here is neither safety nor security critical. Nor are timing aspects a major design issue, since our system's behavior in this regard has long been well understood. Indeed, it have not been narrow timing errors that have been the primary source of our unforeseen software costs in the past, but rather an overabundance of excessively complicated product design features – resulting from a tendency to relegate the treatment of myriad design details to a project's implementation phase – and the collateral obstacles to completely exposing a system's innermost software logic, that have caused difficulty.

This effort is carried out in a commercial setting where the VDM technology has been applied for cost efficiency reasons. The use of VDM in the SIC2000 project can be considered "lightweight" in the sense that no formal refinement or proof has been performed. Instead, in the course of the project, the VDM technology became the focal point of the entire development process, providing a unified treatment of analysis, design, documentation, and testing. Thus a pragmatic approach was used to obtain a precise and abstract model of the most important aspects of the system. We believe that the use of VDM in this project will result in a significant reduction in software maintenance and modification costs during the lifetime of new generations of banknote processing systems.

After this introduction Section 2 gives a short overview of the general aspects of banknote processing systems. Then Section 3 provides an overview of the SIC2000 application. Section 4 describe the VDM technology shortly. Afterwards Section 5 presents the main lessons learned from this work. Finally Section 6 contains a number of concluding remarks and visions for the future use of this kind of technology.

## 2 Software Aspects of Banknote Processing

A banknote processing system (BPS) presents a heterogeneous computing environment, typically including several communication buses and numerous embedded processors which are subject to hard real-time constraints and which control and monitor banknote feature measurement sensors, and banknote transport, stacking, banding, and destruction activities.

Rapid changes in banknote printing technology, in sensor hardware, and the large variety in customer requirements and in their banknote stock [6, 1], result in the need to continually adapt and update the sensor software of a given BPS. From this standpoint, each customer literally has a custom-built system.

In the past, the introduction of a new sensor into a BPS, or a change in the system's sensor configuration, or in the sensor capabilities, necessitated an update in the sensor integration controller (SIC) software. The quality and version control tasks attendant to these updates were a constant drain on GAO's development and service resources.

A hallmark of the newer generation of banknote processing machinery is the high degree of configurability of the sensor software, and of the sorting control software which integrates the individual sensor results into a banknote quality classification that eventually determines the fate of the note. The user now has the valuable but potentially hazardous capability of defining his own banknote sorting logic, which can be loaded into the SIC at runtime. The invariants of these user configurable software entities in the SIC must be described so clearly and completely that there can be no possibility of undetected manipulation, error, or misunderstanding. The ease and reliability with which the integration and sorting control software can be individually configured to meet the processing requirements of each customer influences to a significant extent the adaptation and maintenance costs that accrue during the lifetime of a machine.

Consequently, the GAO recently initiated the SIC2000 software project with the goal of enabling the customer to integrate new sensors into his BPS, or to reconfigure its sorting behavior, without extensive technical support from the GAO. Prior to this project, we had experience with more common development methodologies such as Structured Analysis/Design and UML, and believed they had, on the basis of their syntactic informality, proven themselves to be inadequate to the task of designing a distributed embedded system. Because of the complex and strategic nature of the project, and due to a persistent emphasis in the sensor development department on the importance of software quality issues, the decision was made to develop the software with the aid of VDM.

**Fig. 1.** The BPS1000 banknote processing machine.

## 3 The Application

### 3.1 The Context of the Sensor Integration Controller

The sensor subsystem of the BPS1000 (see figure 1), the banknote processing machine for which the SIC2000 project was originally conceived, consists of up to a dozen optoelectronic sensors, which are mounted along a measurement gallery through which the notes are conveyed by transport belts at a rate of 20 to 30 notes per second. The sensors are responsible for measuring banknote features such as denomination and format; for determining the quality of the notes by checking for soil, graffiti, dog-ears, and other irregularites; for monitoring aspects such as orientation and multiple item events (when two or more notes overlap, making it impossible to accurately measure them) that are related to the transport of the notes through the gallery; and for ensuring the authenticity of the banknotes.

In a working session, the BPS1000 operator can select among several of his predefined sorting modes, thereby defining the context in which banknotes are to be processed. The entity that represents the sorting mode to the SIC is the

so-called Sorting Control Table (SCT), which can be constructed and edited directly on the BPS1000 during an idle mode configuration procedure. The SCT contains identifiers for the properties, and categorization rules that enable the SIC to map a banknote's collection of measured property values into the correct sorting class. The SCTs are stored externally; when the user selects a sorting mode, the corresponding SCT is loaded as a data table into the SIC.

During operation, when a banknote is injected into the measurement gallery, a series of photodetectors monitor its progress, providing the sensors with periodic updates on the note's current position. After a sensor has captured raw measurement data, it transports the data to its evaluation processor, which computes the values of the various banknote properties that the sensor is capable of detecting, and then communicates the results to the SIC via a controller area network.

The SIC accumulates the property values from the sensors, and when the banknote reaches a predefined point near the end of the transport gallery, the SIC uses the SCT to compute the sorting class of the note, transmitting the result to the transport control. Depending on the sorting class, and on the interpretation of that class in the currently selected sorting mode, the transport control routes the note to one of several compartments, where the banknotes are either stacked and banded, or destroyed.

During banknote sorting, each sensor must gather raw data for the note in its field of view while transporting data from previous notes to its evaluation processor. Simultaneously, remote components are delivering data they have measured which contain information that may be needed to parametrize the sensor's measurement and evaluation algorithms on a per note basis, while the transport control is signaling the impending arrival of subsequent notes. Thus, the SIC and the evaluation processors exhibit a high degree of concurrency. Due to the overhead caused by communication and the management of computing resources, and depending on the speed of the machine, the SIC and the evaluation processors can each dedicate no more than 20 to 40 milliseconds of evaluation time to each note. Any deviation from the timing constraint manifests itself immediately in an increased banknote reject rate, much to the customer's dissatisfaction.

### 3.2 The Aims of the SIC2000 Project

Initial versions of the BPS1000 SIC had given the user the capability of defining his own sorting logic in the context of a fixed collection of sensors and banknote properties. In essence, each sensor and each property had to be known to the SIC at compile time.

During the early field-life of the machine, however, it became clear that customers were taxing to a premium the GAO's ability to rapidly integrate new properties and sensors, in many cases sensors manufactured by independent suppliers. It was not difficult to accommodate the customer's reconfiguration wishes in any given instance, but the volume of change and update requests, along with the increasing difficulty of retooling the DB and GUI interfaces to each new SIC version, were becoming a problem that could no longer be ignored. Accordingly,

the GAO sensor department founded the SIC2000 project on the concept of the "Transparent Property Identifier", a design primitive intended to facilitate the anonymous integration of sensors and properties.

The primary requirement on the Transparent Property Id (tpId) methodology was that it should enable the SIC to process any combination of sensors and banknote properties without compile time knowledge of their existence, structure, or meaning. Any quantity delivered by a sensor, on which a banknote can be sorted, should be bound to a tpId, which will then have system-wide validity for its host BPS. One of the main clients of the SIC2000 development, the GAO adaptation team, also demanded the capability of combining individual sensor properties into so-called "compound" properties, which are logical predicates and arithmetical expressions formed from the immediate sensor properties, but otherwise on an equal footing with them. The most important technical constraint on the development would be the minimization of the runtime communication load: the ratio of banknote property data to communication control data on the sensor network bus should be as large as possible.

## 4 The IFAD VDM-SL Toolbox

Although many different sensor components and configuration utilities must operate on the property identifiers, there is no centralized administration of banknote properties or identifiers in the machine, because the individual components are intended to be held as loosely coupled to one another as possible. But this independence carries the possibility of varying syntactic or semantic interpretations of correct property id handling by the different components and utilities. To ensure that the transparent property id methodology functions properly under all circumstances, we specified and modelled the relevant aspects of the system in VDM and checked the specification using the VDM-SL Toolbox.

The IFAD provided a one week training course in VDM-SL (Vienna Development Method – Specification Language) [7, 4] for several of GAO's software engineers, after which they were proficient in the use of the VDM-SL Toolbox, an IFAD product which supports the development of VDM specifications, as well as providing testing and documentation facilities.

The Toolbox is an integrated development environment which has a comfortable familiarity for developers who are accustomed to working in a debug setting. Its interpreter can process test suites formulated in VDM, a feature that we used extensively to gauge the viability and completeness of our specification, which by project's end comprised on the order of 150 pages of VDM-SL, spread across multiple code and test modules.

The Toolbox has an API which enables it to also be used as a Corba server. During integration testing of the SIC2000, we gathered event protocols from the actual components, translated them into VDM syntax, and replayed them in a WinNT process that was implemented as a Corba client. We also built NT front ends to represent the distributed components relevant to the BPS1000 sensor system and connected them to the client via named pipes on networked work-

stations. In this way we were able to replay the protocols through the Toolbox and follow the isomorphic flow of activity in the VDM interpreter. Simultaneously we could see the protocolled activity playing "live" in the front end processes, along with parallel commentary from the specification as it was being interpreted.

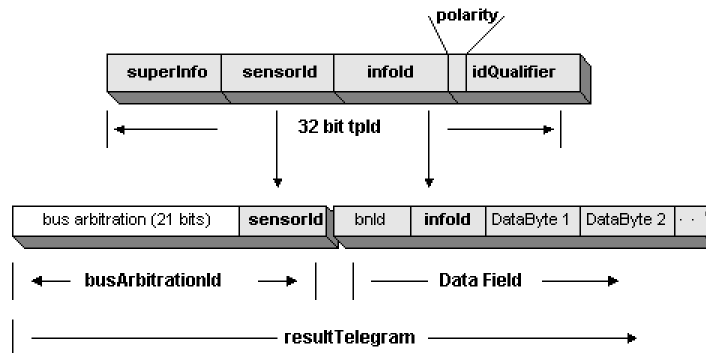


**Fig. 2.** Overview of the Transparent Property Identifier layout.

## 5 Overview of the Development

The first design efforts derived from the requirements on the Tranparent Property Ids described in Section 3.2 were initially formulated in a series of natural text documents, which were primarily concerned with the structure of the tpId (see figure 2). The details were centered around the specific nature of the sensor communication bus. This is a controller area network (CAN), in which each data packet consists of an 8 byte user data field, preceded by a bus arbitration Id. Most of the arbitration Id already had a pre-assigned functionality, but it was decided that a small portion could be set aside for use in the result telegrams in which the sensors transmit their measured property values to the SIC. That portion – used to identify the hardware type of the sensor delivering the result – would be one component of the tpId.

Additionally, one byte in the user data field would form a further component, the *infoId*, of the tpId. The infoId component enables a sensor to individually "tag" each of the several result telegrams that it may transmit. A second byte in the user data is traditionally taken to identify the transport object (banknote) to which the telegram refers, leaving a total of six bytes in the user data field that can be used to transport property values.

At this point, the design ran into complications for the following reason: Tests indicated that the SIC application could reliably process on the order of

2000 CAN telegrams each second. In a typical BPS1000 sorting mode, it may be necessary to evaluate up to 250 banknote properties for each banknote. In a machine that transports 30 notes per second, this amounts to an expected load of about 7500 properties per second. If each property were to be sent in a separate CAN telegram, the SIC would not be able to handle the processing load.

Clearly, as a first step, a sensor should pack as many property results as possible into the available six user data bytes in each of its result telegrams, but it is not immediately obvious how this can be done in a "type-safe" fashion, i.e., in such a way that the SIC can recognize that one sensor intends its data field to be interpreted as a sequence of three unsigned 16-bit quantities, while another sensor may intend its data field to be interpreted as a sequence of six signed 8-bit values. One could attempt to implicitly encode a type cast in the infoId component of the tpId structure to enforce a distinction within a predefined collection of possible interpretations of the data field, but this would not enable the ambient system to reference each property individually.

The designers solved this problem by introducing a phantom component, the *idQualifier*, into the tpId structure. The value of the idQualifier component remains unknown to the sensor, and appears nowhere on the communication buses. Instead it enables a "late binding" capability in the property processing, which can be understood as a context-dependent interpretation of the bus data, the context being defined by a collection of tpId objects in the Sorting Control Table.

### 5.1 Modeling the System

With the addition of the idQualifier component, the data type *transPropId* of the Transparent Property Id can be defined as:

$$\begin{aligned} \mathit{transPropId} :: \; & \mathit{idQualifier} : \mathbb{N} \\ & \mathit{polarity} : \mathrm{SIGNED} \mid \mathrm{UNSIGNED} \\ & \mathit{infoId} : \mathbb{N} \\ & \mathit{sensorId} : \mathbb{N} \\ & \mathit{superInfo} : \mathrm{BUS} \mid \mathrm{COMPOUND} \mid \ldots \; ; \end{aligned}$$

The definition of the type *transPropId* alone is obviously not sufficient for an understanding of its usage. It is necessary to see the transparent property ids "in action" in order to grasp their pragmatics, to interpret and use them. The action that is of interest here is the interaction between the sensor measurement traffic on the communication bus and the information in a Sorting Control Table, which defines the process by which bus traffic is to be transformed into numerical values that can be associated to banknote properties. We provide here an excerpt from the model of the traffic on the communication bus, and follow the resolution of the traffic into values of transparent property ids.

A measurement result telegram from a sensor can be modeled as

$$resultTelegram :: idCAN : busArbitrationId$$
$$bnId : \mathbb{N}$$
$$infoId : \mathbb{N}$$
$$measurementData : abstractByte^* ;$$

The nature of the type *busArbitrationId* is not important; it is enough that there is a well-defined rule for extracting the *sensorId* component of a transparent property id from it (see figure 2). In this definition, an object of type *abstractByte* is any natural number less than 256, and the measurement data is simply a nonempty sequence of such bytes.

**Specifying Arithmetic** An unexpected aspect of the formal notation appears for the first time here in the specification of sensor measurement data as a sequence of *abstractByte*. In the BPS1000 environment there are several processor and compiler dependent versions of the size and memory layout of basic data types such as `byte`, `short`, and `long`. The SIC2000 specification has ramifications for many of the processes in the machine that are external to the SIC, as well as for numerous offline configuration and adaptation utilities. The largest part of the SIC's job is to mediate between the different processes, and to provide an interpreter for the products of the offline utilities. Thus it is imperative that there be no confusion about the alignment, byte order, or bit width of the data types, nor about the location of the data objects in the streams that are transmitted between processes. Since VDM-SL has no predefined concept of memory layout or bit patterns, these low-level details have to be defined from scratch in the specification.

For example, the value of a non-boolean property is determined by the bit width and sign of its type, as well as by the location and nature of the data in the *measurementData* component of a corresponding *resultTelegram*. The construction of the numerical value of a property from a *resultTelegram* is defined by

$$compPropVal : propValLng \times propValPol \times \mathbb{N} \times abstractByte^* \to \mathbb{Z}$$

$$compPropVal\,(pVL, pVP, offset, seqBytes) \triangleq$$
$$\mathsf{let}\ length = byteLngFromPropValLng\,(pVL)\ \mathsf{in}$$
$$\mathsf{let}\ byteBag = [seqBytes\,(offset + i) \mid i \in \{1, \ldots, length\}]\ \mathsf{in}$$
$$\mathsf{let}\ rawValue = convertBytesToNumber\,(byteBag)\ \mathsf{in}$$
$$\mathsf{cases\ mk\text{-}}(pVL, pVP) :$$
$$\mathsf{mk\text{-}}(\mathrm{BIT8}, \mathrm{SIGNED}) \to \mathsf{if}\ rawValue < BYTE1_MODULUS/2\ \mathsf{then}\ rawValue\ \mathsf{else}\ rawValue - BYTE1_MODULUS$$
$$\ldots,$$
$$\mathsf{mk\text{-}}(\mathrm{BIT32}, \mathrm{UNSIGNED}) \to rawValue$$
$$\mathsf{end}$$
$$\mathsf{pre}\ \ offset + byteLngFromPropValLng\,(pVL) \le \mathsf{len}\ seqBytes\ ;$$

The function *compPropVal* takes a sequence *seqBytes* of bytes, and an offset into that sequence, and returns the number whose digits (to the base 256) are $seqBytes(offset + 1)$, $seqBytes(offset + 2)$, etc., where the bit width is indicated by the parameter $pVL$, and the sign by $pVP$. The subroutine *convertBytesToNumber* first computes the property value as if it were an unsigned quantity, and *compPropVal* performs two's complement arithmetic on that result to provide the correct final value.

At first we found it irritating to have to manually specify type casts of the measurement data. When the SIC2000 project came to integration testing, however, the utility modules that were not directly developed from the formal specification produced a number of errors that were caused by a misunderstanding of the rules by which compilers and processors deal with signed quantities and type casting. In contrast, the modules that were directly derived from the formal specification had no difficulty whatsoever. The abstract VDM-SL notation forced us to closely consider and explicitly specify matters seemingly so well understood as two's complement arithmetic and the representation of numbers by bit patterns in memory. As a result, we were able to rapidly localize related errors in the project test phase, and we are now confident that the misunderstanding of computer arithmetic has been effectively banned as a source of error and maintenance costs from our machine.

**Telegram Reception** In the actual implementation of the BPS1000, the communication controller hardware will generate an interrupt at the SIC's CPU whenever it detects that a sensor has transmitted a result telegram on the bus. The corresponding interrupt service routine writes the contents of the telegram to an internal RAM buffer, and then executes a deferred procedure call to a low-priority interrupt. That interrupt's service routine then signals a user level task, which finally processes the contents of the result telegram when the operating system allows it to run. From a design standpoint, these implementation details are superfluous. The sequence of low-level context switches that are set in motion in the SIC by the reception of a result telegram are irrelevant to an understanding of the matter being specified, namely the strategy by which bit patterns generated by sensors can be transformed into numbers associated with banknote properties.

A model of the telegram reception *event* is valuable nonetheless. It allows the reader of the specification to easily establish a connection between the implementation code and the design intent. It also simplifies construction of test suites that can be used to check the appropriateness of the specified data transformations and to support live demonstrations of the specification. The activities that are to be executed upon the reception of a result telegram can be subsumed into a single operation :

$$processResultTelegram : resultTelegram \xrightarrow{o} ()$$

$$processResultTelegram\ (rT) \triangleq$$
$$\quad \textsf{for all}\ p \in \textsf{elems}\ properties$$

```
do if propertyIdMatchesTelegram (p.id, rT)
   then let lSeq = propIdToPropIdGrp (p.id, pGroupV2) in
        let iValue = getPropertyValue (lSeq, p.id, rT.results) in
        processPropertyValue(rT.bnId, iValue, p.id)
```

Whenever a result telegram is received from a sensor, the operation *processResultTelegram* identifies those banknote property ids (listed in the *properties* component of the currently loaded Sorting Control Table) whose values are carried by the telegram, computes their values, and then processes the results of the value computations.

The specification of the operation looks much like conventional program code, which is not surprising because it is, after all, machine executable. The important difference between the VDM-SL specification of *processResultTelegram* and its implementation as a C function with the signature

```
void processResultTelegram(resultTelegram *)
```

lies in the greater narrative density of the VDM notation.

For example, a surprisingly large portion of the production code for the VDM operation *processResultTelegram* is devoted to the implementation of the lines

```
for all p ∈ elems properties
do if propertyIdMatchesTelegram(p.id, rT) ...
```

The implementation of these two lines occupies almost 4000 lines of high level language code spread among 15 different modules, and the corresponding documentation consists of a dozen further pages of natural language text. The discrepancy between the length of this portion of the specification and the size of its implementation is derived from the performance requirements on the target machine code. During banknote processing, the SIC must be able to access up to 375000 entries per second in the *properties* component of a Sorting Control Table, a load that is manageable only if the implementation is optimized for speed. An explanation of the technical handwork that is necessary to effect the code optimization, though useful for maintenance purposes, contributes nothing to an understanding of the system design and can be safely relegated to the documentation of the implementation.

Conversely, the module in which the Transparent Property Id methodology is specified consists of 40 pages of VDM-SL notation, most of which serves to set the stage for defining and testing the operation *processResultTelegram.*

### 5.2 Invariants

The Sorting Control Tables are generated independently of the SIC, often by service or customer personnel, and form the basis for several diagnostic and reporting utilities. Since an improperly formatted SCT can lead to faulty banknote sorting, or unexpected behavior in any of a number of software units, it is imperative that there exist an automatic mechanism that can guarantee the

validity of an SCT before it is operated upon by any of the processes that depend on it. The invariants of the global states of the individual VDM modules that comprise the SIC2000 specification provide the mechanism.

The Transparent Property Id methodology is specified in a VDM module whose global state is *transparentProperties*, which collects the persistent data of the module into a single entity. The invariants of *transparentProperties* define the necessary and sufficient conditions that the property identifier components of an SCT must satisfy. Global state invariants in the remaining modules of the specification define such conditions for other components of the SCT.

For example, the *transparency* of the late property binding mechanism, which is the salient feature of the entire specification, is guaranteed to function properly only if the invariant

$$
\begin{array}{l}
\forall\, p, q \in \mathsf{elems}\ prp \cdot \\
\quad ((p.id.superInfo = \mathrm{BUS} \wedge q.id.superInfo = \mathrm{BUS} \wedge \\
\quad\ \ p.id.infoId = q.id.infoId \wedge \\
\quad\ \ p.id.sensorId = q.id.sensorId) \ \Rightarrow \\
\quad\ (\exists\, x \in qG \cdot \\
\qquad (p.id.idQualifier \in \mathsf{elems}\ x \wedge q.id.idQualifier \in \mathsf{elems}\ x)))
\end{array}
$$

is observed. In this excerpt, the quantity *prp* models the sequence of property identifiers in a Sorting Control Table, and *qG* is a unique, globally defined set of sequences of allowable *idQualifier* components, such that any two distinct sequences in it have no elements in common. The invariant requires that any two measurable (i.e., *BUS*) property identifiers in the same SCT that have identical values in their *infoId* and *sensorId* components are such that the values of their *idQualifier* components lie in the same constituent sequence of *qG*.

Once the invariant has been precisely formulated, it is not difficult to recast its content in colloquial text, which is important for documentation purposes. But clearly the natural formulation of the invariant is in the formal notation, and it is questionable whether it is even feasible to express a complete collection of invariants in a notation with a less formal character.

Altogether the SIC2000 specification defines 27 conjuncts that are present in the overall invariant that an SCT must satisfy. The invariant acted as a requirements specification for a small utility program, which we use to check the validity of an SCT. The utility has been available since the latter stages of the design process, and it performs valuable service as a support tool for developers of other SCT-related units, because it monitors and guarantees the integrity of their interfaces to the SIC. The checker delivers immediate and precise feedback on the correctness of an SCT. In the event of an error, it describes the invariant conjunct that has been violated, and provides the address of the offending position in the SCT. Most importantly, it significantly reduces the innumerable hours of debugging activity – in the laboratory and at customer sites around the world – that developers and service engineers often must perform to compensate for ambiguous or incomplete interface specifications.

## 6 Summary and Perspectives

The development of the formal specification of the SIC2000 project was performed in 1 man-year. The implementation (in the SIC) of the specification was completed in 3 man-months; the implementation of changes in the collection of tools affected by the specification is still in progress. Modular testing of the SIC was finished in 4 man-weeks: several errors were detected, all but one attributable to an imperfect translation of the specification into code. In the case of one error, the specification had to be revised.

The integration testing of the BPS1000 machines that are equipped with the SIC2000 is currently in its final stages. No errors have been detected in the components derived from the formal specification; no change requests have been filed.

The Transparent Property Id methodology has already been or will soon be ported to three further banknote processing systems. One of these systems is similar to the BPS1000; the others have a completely different hardware architecture, along with a different set of CPUs and communication buses. The SIC2000 specification need not be modified to account for the new architectures; the high level language implementations are only moderately different.

We believe that the primary advantage of the VDM technology is the support it provides for the construction of precise and realistic software system models. For complex industrial projects, this is a capability whose value can hardly be overestimated. The price that must be paid for the capability – formulating design ideas in a formal notation, following the implications of design features to their logical conclusions, exposing and codifying all design assumptions in a collection of formal invariants – is not excessively high, considering the alternatives.

Of course, we could have designed the SIC2000 project without VDM. The result would have been a different, a weaker, and eventually a more expensive final product than we were able to produce by using lightweight formal methods. We are convinced that a great deal of our previous software maintenance costs could have been avoided if each of the many software component interfaces had been guarded by an appropriately suspicious set of formal invariants. By building a model of our system, before implementing the system, we were able to accurately assess the consequences of our design for the SIC, and for the many components upstream and downstream from the SIC. Accordingly, we were able to perform on a desktop model a large portion of those troubleshooting activities that are traditionally performed on a system implementation during integration testing or in the field.

The formal specification itself is assuming the status of a long-term storage receptacle of software value. The sensor department at the GAO is now working with IFAD to introduce VDM as an integral part of its software development process.

## Acknowledgement

We would like to thank Sten Agerholm, Paul Mukherjeee and Oliver Oppitz for their comments on earlier version of this paper. In addition Paul Smith would like to thank Dieter Stein for his support.

## References

1. Federal Reserve Board. Foreign central banks. http://www.federalreserve.gov/centralbanks.html.
2. Giesecke & Devrient. Banknote processing systems. http://www.gdm.de/products/noteproc/bps/en/index.html.
3. René Elmstrøm, Peter Gorm Larsen, and Poul Bøgh Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994.
4. John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.
5. Paul Mukherjee. Computer-aided Validation of Formal Specifications. *Software Engineering Journal*, pages 133–140, July 1995.
6. Bureau of Engraving and Printing. The currency. http://www.moneyfactory.com/currency/index.cfm.
7. P. G. Larsen and B. S. Hansen and H. Brunn N. Plat and H. Toetenel and D. J. Andrews and J. Dawes and G. Parkin and others. Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language, December 1996.