Modeling accounting systems to support multiple tasks: A progress report

Walter C. Hamscher
Price Waterhouse Technology Centre, 68 Willow Road, Menlo Park, CA 94025
hamscher@tc.pw.com

Abstract

A domain model in SAVILE represents the steps involved in producing and processing financial data in a company, using an ontology appropriate for several reasoning tasks in accounting and auditing. SAVILE is an implemented program that demonstrates the adequacy and appropriateness of this ontology of financial data processing for evaluating internal controls, designing tests, and other audit planning related tasks. This paper discusses the rationale, syntax, semantics, and implementation of the ontology as it stands today.

Motivation

We wish to develop knowledge-based systems for auditors. Auditors issue opinions on the fairness of the financial statements of their clients. Although financial statements consist of numbers that are typically summations of thousands of other numbers, auditors need not worry about the numbers *per se*. An auditor can instead think about *the system that produced the numbers*: financial records, documentation, accounting policies and procedures, accounting personnel training, security and a host of other non-numeric information. Hence, knowledge-based systems to support auditors could benefit from a foundational representation for data processing systems broadly construed to include both manual and computerized processing steps, while simultaneously suppressing details that are of no audit significance.

Making these ontological commitments in a fresh domain and getting them right is always a difficult undertaking; it is a distinct enterprise from selecting the syntactic form of the knowledge. This paper presents a case study of how these ontological commitments are being made in a principled way for a domain alien to most AI practitioners.

Approach and contributions

There are three key ideas behind the ontology and its realization in the SAVILE implementation: supporting simulation, exploiting existing ontologies, and transforming models to perform multiple tasks.

Supporting simulation Auditing tasks are difficult in part because auditors must understand the relationship between a perturbation in an accounting system and its effects, which can be highly indirect because accounting systems are complex. Fortunately, accounting systems are composed of many interacting processing elements that from an audit standpoint are conceptually simple, and this decomposability suggests that a model-based approach is appropriate: the user would build a computational model of the client system to automate the analysis of the effect of local perturbations. Simulation is one of the fundamental analysis techniques from which others can be derived. Thus an analogy with digital circuits composed of many Boolean elements and analyzed through simulation and other techniques is valid, relevant, and useful to keep in mind. Although a model-based approach has been proposed for financial tasks before, the models have generally been of relationships between variables representing financial and microeconomic quantities [Bouwman, 1983, Hart *et al.*, 1986, Bailey *et al.*, 1990, Bridgeland, 1990, Hamscher, 1991a]. SAVILE is one of the first programs to take this approach with an explicit model of data processing systems.

Exploiting existing ontologies A formidable obstacle in this enterprise has been to develop an ontology of appropriate concepts. Indeed, formulating such an ontology in any fresh domain creates a "chicken and egg" problem: the tasks cannot be properly formulated until an ontology exists, but the concepts of interest and appropriate levels of detail in the ontology depend on the tasks. Also, an effective model-based strategy relies on the ability to acquire the models from domain experts, so that it is crucial to use concepts natural to potential model builders whose expertise lies in accounting, not computer science. SAVILE contributes a set of concepts that merges and extends three ontologies for describing accounting systems developed by (and for) accountants and auditors themselves.

Supporting multiple tasks As a practical matter the effort involved in building a model like that in SAVILE must be amortized over multiple tasks. The promise of reusable models is always implicit or explicit in model-based reasoning research, but experience has shown that for computational efficiency, different tasks require different models. For an example, compare representations of digital circuit behavior for diagnosis [Hamscher, 1991b] and for test generation [Shirley, 1986]. SAVILE incorporates this lesson and takes the approach of using one model for interacting with model builders and users, while performing automated transformations from this model into more efficient models prior to performing each task. SAVILE thus demonstrates a practical approach to model reusability grounded in the experience of previous model-based reasoning efforts and this constitutes another contribution.

The SAVILE ontology is a moving target in an ongoing project having a number of different goals. The basic vocabulary is stable, having not changed in months, and forms the majority of this report. There is currently a fully implemented approach to both the task of evaluating internal controls [Hamscher, 1992] and to the planning of audit tests. A model acquisition environment has been defined, with an implementation in progress and close to demonstration. Other tasks such as risk assessment remain to be fleshed out. Each of these tasks and its relationship to the representation will be discussed later.

Representing accounting systems

Accounting systems process accounting data in the form of paper and electronic records, through activities that create, use, alter, and store those records. Examples of records include an invoice, a receiving report, an entry in a database of receivables from customers, and a weekly sales summary. The "trail" that is left behind in the records by these activities is the raw material that auditors work with when executing an audit at the end of the year. A *record* will thus be a central concept in an ontology for modeling accounting systems that is meaningful to auditors.

There are two general properties of accounting systems that suggest other key concepts. First, they tend to preserve information, in the sense that modification or outright destruction of records is not nearly as common as copying and filing records to allow recovery. The concept of a *repository* will refer to a set of records of the same type, often all physically in the same place (a basket, a folder, a receiving dock) but more generally to mean a set of records all in the same processing state.

Second, records supporting separate economic transactions do not generally interact with one another except to be collected together into summaries and so forth. Hence it is usually possible to understand the behavior of the system by postulating a prototypical transaction, following its processing in isolation, and generalizing. Auditors seem to have no difficulty articulating the accounting processing steps that occur in the course of executing and recording a normal transaction between two economic entities, such as ordering, receiving, then paying for goods. The concept of an *action* is something that a person or other agent does to a record, such as creating it, copying it, comparing it to another record, or putting it into a repository. An *activity* is a sequence of related actions performed by a group of related agents; for example, "cash disbursements" is an activity in which invoices are both examined for validity and paid by issuing checks.

Figure 1 shows the activities and repositories in a model (P&P1) of a typical "purchases and payables" system. P&P1 processes purchases of goods from vendors and ensures that the goods are received, payables recorded, and vendors eventually paid. Records flow downward through the diagram. Each gray boxed item in the graph is an activity. Each other item is a repository, with a nearby icon distinguishing paper from electronic media. The **Stores** activity, for example, has two input and two output repositories, each a set of paper purchase orders (On-Order-File, Filled-POs) and a set of electronic receiving reports (Rr2 and Rr3). Each repository is an input of at most one activity, and an output of at most one.

Crucial steps in an accounting system are those that credit and debit the accounts. The repositories in Figure 1 that are marked with "boxed T" icons represent collections of credits *Cr* and debits *Dr*. Different systems interact, and the boundaries of a system are roughly determined by a single principal account; in this example the system P&P1 produces both credits and debits to the Payables account; the "cash" system would produce credits and debits to the Cash account, and so forth.

This set of concepts and the others introduced below merges and extends existing ontologies. It abstracts away many details about the system implementation as computerized and manual steps, to make explicit the information and steps that are relevant for auditing. The primary source was an experimental program called TICOM designed to support a certain class of queries about actions [Bailey *et al.*, 1985, Bailey *et al.*, 1989]. A secondary source was guidance materials called AGS written within Price Waterhouse for the use of audit staff. A third ontology SEADOC used by a different accounting firm has important similarities [Elliott, 1983]. Descriptions in accounting textbooks also use a similar level of description. All of this provides evidence that the basic concepts are sound if not universal.

Syntax

These concepts are embodied in a language SPLAT: SAVILE Programming Language Attributed to TICOM. Syntactically, this is a set of class definitions in a frame language JOSIE [Nado *et al.*, 1991]. Terms meant to denote classes in SPLAT will be written in this font. For example, each of the basic concepts (record, action and so forth) is a class. SPLAT provides nine record classes such as invoice, order, purchase order, and check, each with a set of fields such as amount, vendor, and payee. Bi-

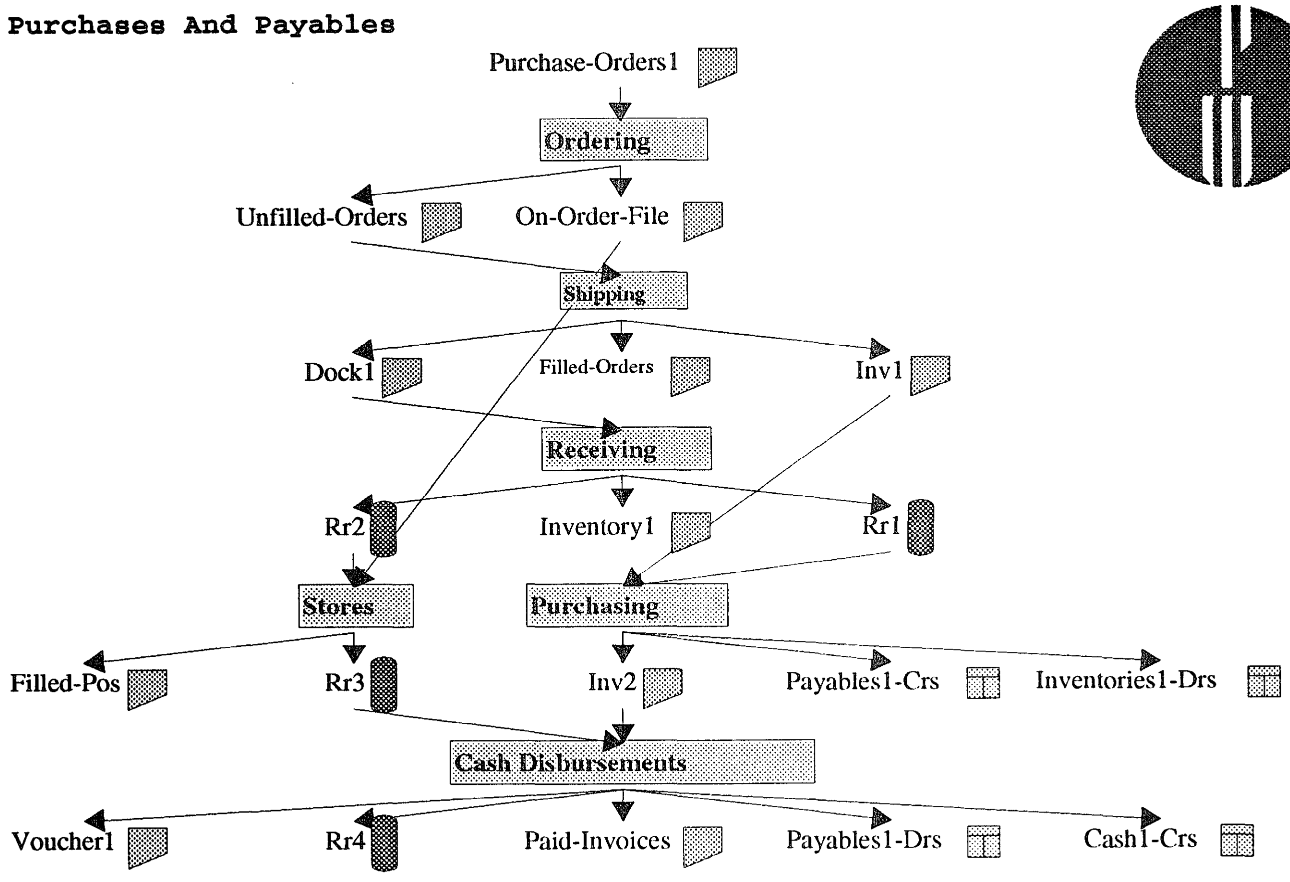

Figure 1: A Purchases and Payables System (P&P1).

nary relationships among classes and individuals, such as the input and output repositories of activity stores, are implemented as slot relations.

Semantics

The semantics of activities and the actions of which they are composed are an elaboration of Petri nets, supporting an interpreter that does event-driven simulation. Records arrive in repositories, and when all of the input repositories of an activity have matching records, the activity fires and executes a sequence of actions. Typically the actions will have the effect of putting records into output repositories, thus firing other activities. This is an elaboration of Petri nets in the sense that activities may consist of complex sequences of actions and the "tokens" are now structured objects (records) that maintain their identity as they change state (move from one repository to another).

Actions

Table 1 shows the current vocabulary of normal actions. Seven of these are inherited from TICOM, the remainder are new. Action sequences are written as programs of an imperative programming language with all statements of the form *action argument**; the arguments are dereferenced when the activity is fired. Dereferencing is context sensitive, in the sense that a ***repository*** argument can specify either the actual repository or the type of record it contains. For example, here is the action sequence of the **Stores** activity:

```
Wait-for Purchase-Order Receiving-Report
Ensure-Equal Purchase-Order Receiving-Report
Put Purchase-Order Filled-POs
Put Receiving-Report Rr3
```

First, a matching purchase order and receiving report arrive at the input repositories (the Wait-for). Next, each pair of fields that the two records have in common is tested for equality, and if any discrepancy occurs it is corrected (the Ensure-Equal). This represents the effect of accounting personnel reviewing and correcting discrepancies. An ensure-equal action is an example of an *internal control*: an action that does not produce any new information itself, but is only intended to detect problems in existing information. Then, the purchase order is put into the repository Filled-POs and the receiving report put into Rr3.

assign *(field destination-record) (field source-record)*
Put the contents of the source record field into the destination record field.

compute *(field destination-record)* &rest *arguments*
Assign to the destination record field a result that depends in some way on the arguments.

create-empty *record record-class* &optional *source-record*
Create a new record of a certain class and indicate that it was derived in some way from the source record.

create *record record-class* &optional *source-record*
Create a new record and assign all of the fields that it has in common with the source record.

copy *source-record record*
Create a new record that is a copy of the source record.

credit or **debit** *record account*
Use the "amount" field of the record to create a new debit or credit record in the account.

post *record debit-account credit-account*
Perform both a credit and debit operation to the appropriate accounts.

ensure-equal *first-record second-record*
Check that all of the fields that the two records have in common agree, and repair the problem if they do not.

get *record repository*
Extract a record from the repository

put *record repository*
Move a record into the repository.

transfer *record activity*
Move a record to another activity by putting it into an intermediate repository.

sequence &rest *actions*
Perform each of the actions.

wait-for-all &rest *repositories*
Get a matching set of records (that is, they have a common source), one from each repository.

monitor-buildup *repository* &rest *other-repositories*
If there is a record in the repository, periodically check the other repositories for matching records, and repair the problem if they do not appear.

wait-for &rest *repositories*
Monitor the buildup of records in all repositories, and wait for a match.

Table 1: SPLAT Action Vocabulary.

Some actions can be "macro expanded" into a sequence of other actions. For example, the action Copy Rr1 Rr2, meaning "make a copy called Rr2 of the record Rr1," expands into a sequence including a create and several assign actions, one for each field in the record Rr1. Actions that do not expand are *primitive*.

Potential mistakes in, omissions from, or deliberate changes to accounting procedures play a key role in the analysis of a system by an auditor. These are called activity *failures* in SAVILE; specifically, each action is associated with one or more possible failures. SAVILE enumerates the set of possible failures in an activity by local substitution in its action sequence; each action is replaced in turn by each of its possible failures. For non-primitive actions, expansion is performed first, and then the failures are enumerated. SAVILE is thus able to enumerate the potential failures in each activity based only on the SPLAT behavior description. This is a good and important property because it means that the auditor does not have to enumerate the possible failures, but need only provide a description of the correct behavior of the system.

Model acquisition

Part of standard audit methodology includes an annual *systems update* in which the audit planner documents the organization of the client accounting systems, or, more typically, updates the documentation prepared the previous year. Existing software to support this includes a flow graphing program which allows the user to draw pictures of repositories and activities and to write narrative descriptions of the actions performed in each activity. However, having produced this documentation, no existing software uses this information about the way the client system is organized to assist in the remaining task of analyzing its audit consequences. The potential benefits of a SAVILE model are thus substantial.

The key difference between what audit planners already do and what SAVILE requires is the formalization of the knowledge about repositories, the records they contain, and the actions taken in each activity. There are several reasons to believe that this additional cost will not present an unacceptable burden. First, the level of abstraction in SPLAT is very high and uses concepts and terminology familiar to anyone with an undergraduate accounting background. Second, SPLAT is highly constrained. For example, given a fixed set of input and output repositories for an activity, the set of meaningful actions is restricted enough that an intelligent editor might be able to construct a reasonable guess at an appropriate action sequence. Third, it is rare to encounter a novel accounting system involving novel record types and activities; most are minor variations of standard types and structures, and a client model could probably be produced substantially via copy and edit. A graphical interactive model acquisition environment is currently under development to see whether these intuitions are right.

Summary of the representation

SAVILE has a foundational representation for data processing systems that exploits existing terminology and uses a level of detail not substantially different from what practitioners already provide. The ontology relies on the fact that accounting systems are composed of many interacting actions that are each simple from an audit viewpoint. The user builds a model of the client system to allow automation of the analysis of the effect of local perturbations in the system, and graphical interaction is always done with respect to this model no matter what the task.

Tasks

SAVILE is oriented toward audit planning (hence the name: detailed audit plans are, in audit jargon, "tailored" to the client, as done by the pricey London haberdashery Savile Row). A strategy for audit planning can be formulated in SAVILE as the following series of subtasks:

Assessing risk Some failures are more likely than others, and the auditor must ensure that the plan addresses them. The risk of failure for a given action depends on factors such as its complexity, whether the data used change frequently, whether it is manual or automated, and so forth. Factors that might increase the risk of a given failure can already be systematically generated from SPLAT. These factors also have relationships to one another that are outside the current SAVILE ontology; knowledge-based risk assessment systems already exist to provide a basis for this extension [Dhar *et al.*, 1988, Boritz *et al.*, 1989, Peters, 1990].

Significance Failures vary in their audit significance. Some have virtually no audit significance and can be filtered out of further consideration; for example, purchase orders that never get sent to a vendor do not make the financial statements incorrect. Conversely, a failure to post a credit to Payables would result in the final balance being understated, which could have audit significance depending on the total amount involved. In the P&P1 example, there are 40 possible failures but only 28 are significant under this criterion. In the causal network derived for the control evaluation task (discussed below) the impact on the financial statements of each failure is found by examining the set of repositories reachable from it in the causal network. This is a useful qualitative filter, and including the reported year end balance in each account would permit fine grained judgements as to relative significance.

Evaluating controls Having focused on likely failures with audit significance, some can be filtered out of consideration because they are detected by internal controls. Every accounting system contains controls that test for mistakes in previous processing steps; the problem is to establish whether a given failure is detected by any control. In the P&P1 example, of the 28 significant failures only 14 are undetected by controls.

In SAVILE, a failure causes either a loss of records or propagation of corrupted records; downstream control actions such as ensure-equal and monitor-buildup detect the failure by detecting a discrepancy between the lost or corrupted record and other records. SAVILE enumerates the possible failures from the SPLAT model, inserts the failures, and simulates the movement of typical records through the system. The simulation results are abstracted on an activity-by-activity basis into a causal network that (i) suppresses details about records, fields, and normal execution order and data flow (ii) makes explicit the causal relationships between underlying failures and their symptoms in repositories downstream and (iii) embeds the assumption that any single transaction can be affected by at most one non-control failure (but any number of failing controls can still occur during a single transaction, and many distinct failures may each perturb different transactions). This abstraction reduces the problem of finding the controls that detect a given failure to that of finding a path in a directed graph [Hamscher, 1992].

Design of substantive tests The final audit plan should test the failures surviving the above filters. Each *substantive test* in the plan involves examination of a sample of records for evidence of failures. This is expensive in staff time and should be minimized; however, all testing must be completed within a short period after the end of the year, so planning for coverage of potential failures is equally important.

The initial problem is to design a substantive test capable of detecting instances of a particular failure. For example, suppose that the failure no-put purchase-order filled-pos is to be tested for. Intuitively, one way to test for this is to examine a sample of records that were each expected to be paired with a purchase order; in the P&P1 example, a sample of receiving-reports extracted from repository rr3. These receiving reports would then be traced to their matching purchase orders in filled-pos. To construct this test, SAVILE first transforms the model into dependencies among the fields of records in different repositories. For example, the action copy rr1 rr2 in the **Receiving** activity means that all of the fields in the records of rr2 depend on the fields in records of rr1. This transformation suppresses, among other information, the order of execution of actions. It makes explicit the actions that need to have succeeded in order for the test to succeed, hence it makes explicit the failures that the test could detect. SAVILE then performs a kind of *path sensitization* [Bennetts, 1982] within these dependencies. In path sensitization, the goal of testing a particular action is decomposed into the subgoals of sensitizing the failure (working upstream toward a sample of records that were its "input") and propagating the result (working downstream from a sample of records that were its "output").

Each test designed for each (unfiltered) failure covers a small set of failures. Planning an audit then reduces to finding a set of tests to cover them all. Finding a minimal cover is intractable in principle, but probably acceptable in practice because of the modularity of systems and the ability of path sensitization to produce narrowly focused tests. Finally, many of the substantive tests have steps in common, and for efficiency SAVILE merges them where possible. In the P&P1 example, there are 45 possible tests to cover 14 failures, but six of the tests cover all the failures and SAVILE subsequently merges these six into just two (complex) tests.

Conclusion

SAVILE demonstrates a new model-based approach to support auditing tasks. The foundation of this approach is an ontology for modeling financial data processing systems with ancestry in the literature of domain experts and supporting a plausible model acquisition strategy. SAVILE also suggests a practical approach to reusable models, grounded in the observation that each task may require transformation into an intermediate representation, substantiated by the use of a single model both for control evaluation and audit planning.

Acknowledgments

Maureen McGowan supplied domain expertise, including detailed analysis of the rationale behind real audit plans. Jim Peters and Andrew Bailey pointed me at accounting research literature on internal control evaluation. Bob Nado supported JOSIE. Beau Sheil helped refine many ideas in SAVILE. R. Michael Young implemented the SAVILE user interface and helped bludgeon a reluctant CLIM into producing PostScript output for Figure 1.

References

[Bailey *et al.*, 1985] A. D. Bailey, G. L. Duke, J. Gerlach, C. Ko, R. D. Meservy, and A. B. Whinston. TICOM and the analysis of internal controls. *The Accounting Review*, LX(2):186–201, April 1985.

[Bailey *et al.*, 1989] A. D. Bailey, K. S. Han, R. D. Stansifer, and A. B. Whinston. The advanced internal accounting control model using a logic programming approach. Working Paper of 7/20/89, 1989.

[Bailey *et al.*, 1990] A. D. Bailey, Y. Kiang, B. Kuipers, and A. B. Whinston. Analytical review and qualitative and causal reasoning in auditing. Draft of 4/15, 1990.

[Bennetts, 1982] R. Bennetts. *Introduction to Digital Board Testing.* Crane Russak & Company, New York, 1982.

[Boritz *et al.*, 1989] J. E. Boritz, R. G. Kielstra, and A. M. Albuquerque. A prototype expert system for the assessment of inherent risk and prior probability of error. Report, School of Accountancy, University of Waterloo, Waterloo, Ontario N2L 3G1, February 1989.

[Bouwman, 1983] M. J. Bouwman. Human diagnostic reasoning by computer: An illustration from financial analysis. *Management Science*, 29(6):653–672, June 1983.

[Bridgeland, 1990] D. M. Bridgeland. Three qualitative simulation extensions for supporting economics models. In *Proc. 6th IEEE Conf. on A.I. Applications*, pages 266–273, Santa Barbara, CA, March 1990.

[Dhar *et al.*, 1988] V. Dhar, B. Lewis, and J. Peters. A knowledge-based model of audit risk. *AI Magazine*, 9(3):57–63, Fall 1988.

[Elliott, 1983] R. K. Elliott. Unique audit methods: Peat Marwick International. *Auditing: A Journal of Practice and Theory*, 2(2):1–12, Spring 1983.

[Hamscher, 1991a] W. C. Hamscher. Model-based financial data interpretation. In *1st Int. Conf. on AI Applications on Wall Street*, New York, October 1991. IEEE Computer Society Press. Also in *Working Notes of the 2nd International Workshop on Principles of Diagnosis* (Technical Report RT/DI/91-10-7, Dipartimento di Informatica, Universitá di Torino, 1991).

[Hamscher, 1991b] W. C. Hamscher. Modeling digital circuits for troubleshooting. *Artificial Intelligence*, 51(1-3):223–271, October 1991. Also in J. de Kleer and B. Williams (eds.) *Qualitative Reasoning about Physical Systems II* (North-Holland, Amsterdam, 1991 / MIT Press, Cambridge, Mass., 1992) and in W. C. Hamscher, J. de Kleer and L. Console (eds), *Readings in Model-based Diagnosis* (Morgan Kaufmann, San Mateo, Calif., 1992).

[Hamscher, 1992] W. C. Hamscher. Analysis of accounting systems via abstraction of simulation results into causal networks. Technical Report 25, Price Waterhouse Technology Centre, Menlo Park, CA 94025, January 1992.

[Hart *et al.*, 1986] P. E. Hart, A. Barzilay, and R. O. Duda. Qualitative reasoning for financial assessments: A prospectus. *AI Magazine*, 7(1):62–68, Winter 1986.

[Nado *et al.*, 1991] R. Nado, J. Van Baalen, and R. Fikes. JOSIE: An integration of specialized representation and reasoning tools. *ACM Sigart Bulletin* special issue on implemented knowledge representation and reasoning systems, 2(3):101–107, June 1991.

[Peters, 1990] J. M. Peters. A cognitive computational model of risk hypothesis generation. *Journal of Accounting Research*, 28(Supplement):83–103, 1990.

[Shirley, 1986] M. H. Shirley. Generating tests by exploiting designed behavior. In *Proc. 5th National Conf. on Artificial Intelligence*, Philadelphia, PA, August 1986.