A Domain Specific Property Language For Fraud Detection To Support Agile Specification Development

Aaron Calafato
Dept. of Computer Science
University of Malta
aaron.calafato.06@um.edu.mt

Christian Colombo
Dept. of Computer Science
University of Malta
christian.colombo@um.edu.mt

Gordon J. Pace
Dept. of Computer Science
University of Malta
gordon.pace@um.edu.mt

Abstract—**Fraud detection has long been established as an indispensable element of fiscal reporting. Typically, fraud experts identify patterns which are used by a development team to implement a system to run over existing data for the identification of suspicious behaviour. The process of defining rules is lengthy, expensive and error prone; even for minor variation of properties. In this paper we propose a framework enabling fraud experts to directly experiment with fraud patterns, getting immediate feedback based on past data so as to enable their verification and refinement. Using a domain-specific language, the framework will empower fraud experts to design the patterns, whilst the use of automated fiscal trail analysis enables feedback from the effect of the rules onto the system.**

I. Introduction

Fraud is a problem that can be found in any financial transaction system. The cost of fraud has been reported to be substantial, with for instance, 234 cases known to the police in Malta in the year 2010[1]. In the USA, it was reported that "The IRS could net almost $28 billion from tax fraud and errors that are identified and ripe for collection."[2]. Consequently, focusing on the design of patterns that can be used to detect fraudulent cases of tax reporting, has been an important goal in any financial system.

Fraud detection can be approached in various ways with the most straightforward way being that of manually accessing various documents and auditing any suspicious ones according to the fraud expert's judgement. However, this methodology has two well known disadvantages: patterns which are common between cases can only be defined with the fraud expert's intuition, and that the process is too lengthy.

One approach which tackles these problems is that of using machine learning techniques. Algorithms such as Artificial Neural Networks (ANN) and Genetic Algorithms can learn patterns when provided with historic data of financial transactions [1]–[3]. Rules are then automatically induced by the algorithm using a test data set. In the case of fraud detection for a tax system, the test data could be composed of all recorded tax entries, with the ones confirmed as fraudulent marked accordingly. When a new tax return is submitted in the system, the algorithm can advise whether the tax return is to be audited based on the learnt patterns. However, the algorithm has shown to be unsuccessful without intensive work from the fraud expert and IT team [1]. Furthermore, testing the ANN is not straightforward, since this approach does not depict the learnt rules.

With an ANN, the fraud expert might find it difficult to deduct the source of the problem if the system selects erroneous records. It would be more natural for a fraud expert to define rules in the first place, instead of analysing why the result of an algorithm returned erroneous patterns. This issue calls for a system which can have the rules defined precisely, both at testing and implementation stages.

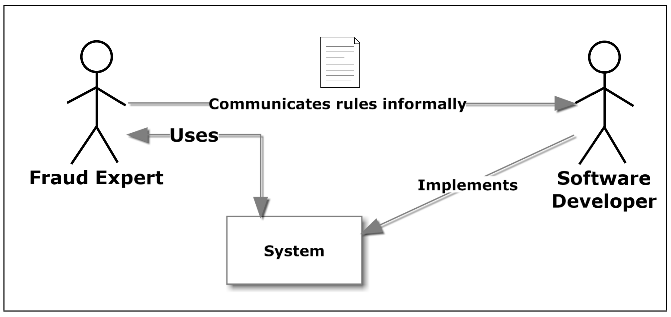

Fig. 1. Approach of manual generation of fraud rules

Having clearly defined rules, thus separating any technology issues (e.g. Why was this case selected?) from the business ones (e.g. Is this a correct fraud pattern?) would enable the fraud expert to focus more on the latter. To date, most of the IT fraud systems implement rules directly in the main system. Figure 1 shows a typical cycle required to adopt new fraud detection system rules. The fraud expert starts off by providing a list of rules to the developer who then implements the system based on his/her interpretation of the rules. The system matches cases that are supposed to be fraudulent, and these are then examined by the fraud expert. Based on the matched cases, the fraud expert may decide to refine the rule, thus inducing an iterative process.

[1]http://www.nso.gov.mt/statdoc/document_file.aspx?id=3173
[2]http://www.taxpolicycenter.org/UploadedPDF/900641.pdf

Certain issues emerge in this approach. The first issue is the disparity between the area of expertise of the fraud expert and the software developer, where such discrepancy may influence the interpretation of the rules. In a taxation system, for example, a taxpayer may be interpreted as anyone who is registered with the tax department or anyone who has ever paid taxes. Due to a possible lack of knowledge of the area, the software developer may neglect such details when implementing the rules, resulting in an inaccurately implemented rule. Additionally, even if the rule has been interpreted correctly, the system may contain a number of bugs. Therefore, severe testing has to be done on the system both when implementing new rules and when refining existing ones. This clearly indicates that the whole cycle may be lengthy, even when dealing with minor changes.

Once cases are identified as fraudulent by pattern matching, these are subsequently audited. The cost of auditing someone may be substantial, so in order to avoid unnecessary expenses, the number of false positives (cases wrongfully identified as fraudulent) should be minimised by analysing the impact of new rules prior to implementation.

Another disadvantage of this approach is maintainability: when a new rules needs to be added, existing ones have to be analysed and understood. This might not always be straightforward, especially with systems implemented years ago and maintained by a number of different developers. Rules which have been in place for a number of years should be documented thoroughly, and this requires significant effort to maintain [4].

It can be argued that it is crucial to involve the fraud expert when defining or discovering rules. Instead of tackling any technical issues, the fraud expert should focus on the rules' descriptions. Subsequently the fraud expert would be aided by a monitoring system which, given the rules, selects cases to be audited. An intermediate process is required to automate certain steps, making the process faster and less dependent on an IT person.

The major contribution in our work will be based on Domain Specific Languages (DSL) to tackle the description of the fraud detection patterns. DSLs [5], [6] are languages which differ from generic languages in two aspects: (i) they contain predefined concepts which are suitable for any issue within the domain, and (ii) they provide a more restricted set of expressions used to define rules. The first advantage is crucial for fast definition of rules without the chance of inducing bugs, since concepts like 'income', 'expenses', and 'profit' would be predefined in a formal manner. The fraud expert can then focus on the actual implementation of rules rather than the technicalities that come with the implementation of basic components. The second advantage is the restricted language of a DSL, which when well designed, provides a set of operators concise enough to avoid ambiguities [7] yet expressive enough to define any possible rule within the domain [8].

This process can be achieved using a runtime verification (RV) framework to continuously monitor the system's behaviour, when provided with a set of properties. RV tools are sometimes enhanced by an interface which aids the user, in this case the fraud expert, in defining and refining rules. A framework with a design mechanism and automated RV, will be discussed in more detail in section 2. This approach will be compared to other works related to the domains of DSLs and RV, in section 3. Finally, we will conclude our paper and illustrate the way forward for our proposed work in section 4.

II. Proposed Architecture

Our proposed solution for fraud detection allows for an iterative process. Iterations are needed to define and refine rules without any dependency on IT personnel. As shown in figure II, the process will start from the description of the rules through a specific language for fraud detection, thus allowing for a more formal, less ambiguous set of rules. These rules will be automatically compiled and processed vis-à-vis the main system. The result of this processing will be returned to the user. The concise language and results may be augmented into a workbench aiding the fraud expert by abstracting the excessive technicalities found in the automated process.

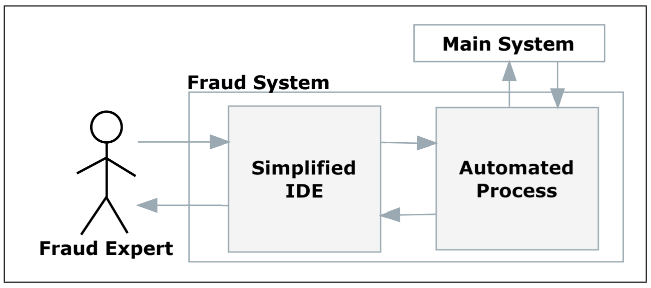

Fig. 2. Architecture of automated system

A. A concise but expressive language

Our proposed system has to start off with a simple IDE (Integrated Development Environment) capable of defining rules in a clear and manageable way, such as a Domain Specific Language. The use of the DSL paradigm has been valuable in a wide range of domains, from graphics design [6] to controlled language for contracts [7]. DSLs strive to provide the expressiveness needed to describe the desired properties, which, in terms of the fraud domain, refers to the fraud detection rules to be composed.

A DSL has the advantage of being more concise than standard programming languages, by providing the user with the expressivity only for the concerned domain. For instance, an example of an informal rule would be:

"Anyone declaring less than 2,000 Euro per year for three years is likely to be fraudulent."

It can be argued that this rule contains a number of unspecified details:

- Does "Anyone" refer to any taxpayer category (individuals, companies, etc.)?
- Are the 2,000 Euro income or profit?
- Are the three years consecutive or not?

A DSL aids in eliminating any ambiguities with the use of predefined operators. For instance, the "Anyone" keyword in the first question above may be too generic, and formally predefined constructs like "Taxpayer" or "Employee" would reduce such ambiguity. Furthermore, in practice, the second question may have a significant effect on the cases detected for fraud. Assuming that the 2,000 Euro refers to income, companies declaring more expenses than income for a number of years would not be flagged, since the above rule targets the income and not the profit. As a final result, companies never paying taxes, due to the lack of profit, would end up not being audited even though these may be very suspicious.

With regards to whether the three years are consecutive or not, changing this detail in a manual system would require major maintenance cost for implementation and testing. This is because definitions of temporal logic can be hard to define in an imperative language. Whilst it may be straightforward to define any years without order, it would be more intricate to define the "consecutive" operator. On the other hand, constructing the "consecutive" operator in a DSL would remove this heavy maintenance cost. As a result of these three questions, a more formal rule may be:

"Any company declaring a profit of less than 2,000 Euro per year for three consecutive years is likely to be fraudulent."

A disadvantage claimed for DSLs is the initial cost [9], which is higher than that of a standard application having the rules hard-wired in it. However, fraud detection patterns have been proven to change from time to time [2], thus inducing maintenance costs. When compared to manually implementing the rules into the system, DSLs require less maintenance since a number of core concepts are predefined in the system, such as the "consecutive" operator, since the rules are dynamically imported into the system. Thus, any effort in the refinement of rules is transferred to the fraud expert who can independently describe rules as shown in the following sections.

B. *Automated monitors*

Once the rules are defined, rules have to be compiled and processed against the main system. Automated interpretation and compilation of the rules reduces the chances of software bugs, especially when there is an increase in the number of iterations needed to refine a rule. Maintenance is rigorously reduced by removing the dependency on the IT person. This will be done by automating a number of steps:

1) Compiling the rules into an intermediate structure
2) Generating a number of monitors
3) Executing monitors on a subset of the system trace

With automated execution of the rules, the system may execute the monitors on the existing logs and compare them to the previous set of results, thus providing results directly to the fraud expert in a more consistent manner.

C. *Feedback Mechanism*

Automated monitors allow for timely feedback when refining rules. However, in order to achieve a feedback mechanism, a monitoring system has to simulate the rules on a set of data. This is not possible at runtime, since online monitoring examines the current trail of events, whilst feedback of a rule refinement would require the simulation of the rule on an extensive trail of events in a short period of time. This can be done using offline monitors [10], which allows the monitoring system to retrieve the log of events from the existing financial system. The feedback mechanism would have the power to define the range of events to examine, since a rule might have to be checked on a limited span of events.

The fraud expert would interact with the system by designing rules and analysing the feedback from the rules. Having this kind of behaviour in a timely manner would ultimately allow for fast rule refinement and sensitivity analysis, before rules are permanently applied onto the system.

Once the feedback is returned to the user, it would be ideal to categorise the selected cases according to the respective rule with which these were identified. In order for this to be achieved, the automation discussed previously has to be done in both ways: (i) from the DSL to the monitors and (ii) from the monitors' results to the respective rule. Therefore, besides the compilation of monitors, a backward mapping has to exist from the compiled monitors to the rules described in the DSL. Figure 3 shows how the rules written with the DSL are automatically compiled and, using a monitoring system, cases to be audited are returned to the domain expert.

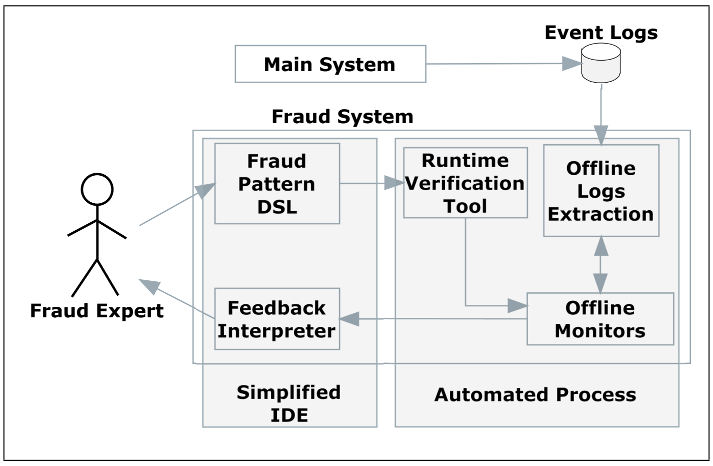

Fig. 3. Approach of automated system

III. Related Work

Within this domain, research on the Artificial Intelligence field [1]–[3] tackles rules' discovery and/or optimisation through machine learning using algorithms like Artificial Neural Networks (ANN) and Genetic Algorithms. Whilst claiming that, following an amount of work this approach may be effective, the works in [1], [3] show that it is also noted that the fraud expert has to analyse the results of the machine learning algorithm in a more technical manner. Technologies like the ANN do not provide adequate feedback with regards to why certain cases have been selected and, in a sensitive area like fraud, such analysis is vital.

Our two major areas of interest in this work are domain specific languages and monitoring systems, which when combined, provide the feedback required to refine rules. On the topic of specific languages, [11] shows how graphical representation can abstract processes from the excessive technicalities. Graphical aid is advantageous when compared to scripted approaches, since syntax errors cannot occur as the input is controlled by the designated tools. In the domain of fraud detection, it would be useful to have a descriptive language which can be easily documented but this cannot be directly achieved solely through graphical tools. Perhaps, a suitable approach may be to have multiple compilations, from the specification of rules to the outputted monitors and documentation, such as the work found in [12].

Two advantages of DSLs found in [7]–[9], [13] highlight how DSLs are suitable to define rules in a clear and concise manner. [7] shows how DSLs may be used to reduce ambiguities with a well defined structure for contracts. The second advantage is the predefined core concepts found in DSLs like [8]. This work starts off from the very basic operators of the domain and builds up to the more complex operators based on the basic ones. The final result is a concise yet expressive enough set of operators. The conciseness of DSLs is also highlighted in Lula [13] which uses a DSL to describe how to define stage lighting. Being so concise, Lula does not cater for temporal logic, however this can be augmented through Lulal, another DSL found in the same work. This approach may be useful in our domain especially with a large number of categories for taxpayers (self employed, employed, companies, etc.).

The work found in [4] is the one that most closely resembles monitoring aspect of our proposed system. The architecture starts off with a set of requirements which, when modelled, are converted into a Compliance Rule Graph (CRG). Whilst the modelling from the requirements to the CRG is unclear, [4] claim that an advantage of a graphical representation is that there is a very simple mapping between the designed rules and the actual structure used to monitor the system. [4] claims that one of the most important factors in a monitoring system is “root cause” analysis, i.e. knowing why the system’s behaviour was found to be invalid. The approach of providing the results of the processing within the same graphical models may be suitable for refining rules.

IV. Conclusions

In this paper, we have proposed a system architecture capable of handling the description of fraud detection rules in a natural manner. This will be done using a DSL as a descriptive tool and a monitoring system which examines the existing trail of log for possible fraudulent cases. A fraud expert will be capable of defining rules with diminished or no assistance from an IT person. Being an iterative process, rules may be refined, whilst a timely feedback will highlight the effect of the refinement. The use of external DSLs is hypothesised to be more suitable for this work, as it provides a more concise yet expressive enough language. In addition to this, a DSL aiding tool would provide further assistance to the fraud expert at the rule-description stage.

In order to aid the fraud expert, a number of steps are to be automated. Rules are to be internally formalised and automatically compiled into monitors, thus shortening the process. To our knowledge, no work tackles the description of rules and the simulation of the rules upon the existing trail of logs. Our contribution will be therefore to combine a well-designed DSL with a feedback mechanism illustrating the effect of rule refinement on the main system. This approach is ideal for a sensitive domain like fraud detection systems where one needs to thoroughly check the effect of a rule before actually auditing cases.

References

[1] S. Maes, K. Tuyls, B. Vanschoenwinkel, and B. Manderick, “Credit card fraud detection using bayesian and neural networks,” in *In: Maciunas RJ, editor. Interactive image-guided neurosurgery. American Association Neurological Surgeons*, 1993, pp. 261–270.

[2] S. Rosset, U. Murad, E. Neumann, Y. Idan, and G. Pinkas, “Discovery of fraud rules for telecommunications-challenges and solutions,” in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 1999, pp. 409–413.

[3] T. Fawcett, Foster, and F. Provost, “Adaptive fraud detection,” *Data Mining and Knowledge Discovery*, vol. 1, pp. 291–316, 1997.

[4] L. T. Ly, S. Rinderle-Ma, D. Knuplesch, and P. Dadam, “Monitoring business process compliance using compliance rule graphs.” in *OTM Conferences (1)*, ser. Lecture Notes in Computer Science, R. Meersman, T. S. Dillon, P. Herrero, A. Kumar, M. Reichert, L. Qing, B. C. Ooi, E. Damiani, D. C. Schmidt, J. White, M. Hauswirth, P. Hitzler, and M. K. Mohania, Eds., vol. 7044. Springer, 2011, pp. 82–99.

[5] M. Fowler, *Domain Specific Languages*, 1st ed. Addison-Wesley Professional, 2010.

[6] J. Bentley, “Programming pearls: little languages,” *Commun. ACM*, vol. 29, no. 8, pp. 711–721, Aug. 1986. [Online]. Available: http://dx.doi.org/10.1145/6424.315691

[7] G. J. Pace and M. Rosner, “A controlled language for the specification of contracts,” in *CNL’09 Proceedings of the 2009 conference on Controlled natural language*, 20-24 June 2009, pp. 226–245.

[8] S. P. Jones, J. M. Eber, and J. Seward, “Composing contracts: an adventure in financial engineering (functional pearl),” in *ICFP ’00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*. New York, NY, USA: ACM, 2000, pp. 280–292. [Online]. Available: http://dx.doi.org/10.1145/351240.351267

[9] P. Hudak, “Modular domain specific languages and tools,” in *Proceedings of Fifth International Conference on Software Reuse*. IEEE Computer Society, Jun. 1998, pp. 134–142.

[10] C. Colombo, G. J. Pace, and P. Abela, “Offline runtime verification with real-time properties: A case study,” in *Proceedings of WICT 2009*, 2009.

[11] M. Carro and M. V. Hermenegildo, “Some design issues in the visualization of constraint logic program execution.” in *APPIA-GULP-PRODE*, J. L. Freire-Nistal, M. Falaschi, and M. V. Ferro, Eds., 1998, pp. 71–86.

[12] A. V. Deursen, “Little languages: Little maintenance?” p. 19, 1998. [Online]. Available: http://homepages.cwi.nl/ arie/papers/domain.pdf

[13] M. Sperber, “Developing a stage lighting system from scratch,” in *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP ’01. New York, NY, USA: ACM, 2001, pp. 122–133. [Online]. Available: http://doi.acm.org/10.1145/507635.507652