A Controlled Natural Language For Tax Fraud Detection

Aaron Calafato, Christian Colombo, and Gordon J. Pace

University of Malta
aaron.calafato.06|christian.colombo|gordon.pace@um.edu.mt

Abstract. Addressing tax fraud has been taken increasingly seriously, but most attempts to uncover it involve the use of human fraud experts to identify and audit suspicious cases. To identify such cases, they come up with patterns which an IT team then implements to extract matching instances. The process, starting from the communication of the patterns to the developers, the debugging of the implemented code, and the refining of the rules, results in a lengthy and error-prone iterative methodology. In this paper, we present a framework where the fraud expert is empowered to independently design tax fraud patterns through a controlled natural language implemented in GF, enabling immediate feedback reported back to the fraud expert. This allows multiple refinements of the rules until optimised, all within a timely manner. The approach has been evaluated by a number of fraud experts working with the Maltese Inland Revenue Department.

1 Introduction

Fraud is a critical aspect in any financial system, not least of which in the area of taxation. In general, fraud can be addressed by identifying rules which can either be automatically discovered or specifically defined. Much work has gone into statistical and machine learning techniques to identify rules or patterns automatically e.g. [MTVM93,TD99], however to date they have proved to be of limited effectiveness. Our focus has been to aid fraud experts in the definition of rules. This process typically consists of (i) human fraud experts identifying patterns of income and/or expenditure which might indicate fraud; (ii) these patterns are explained to an IT team, who implements the necessary code to identify matching individuals and companies from a database; (iii) based on which, the fraud expert may refine the pattern, resubmitting it each time to the IT team. This process, illustrated in Fig. 1 typically goes through many iterations until effective patterns are identified by the fraud expert to extract suspicious cases with acceptable rates of false positives and negatives. The most challenging problem is that there are many points of failure in such a workflow, and when a fraud expert gets results from a submitted pattern which are not as desired, it could be for one (or more) of various reasons, from the developer misunderstanding the informal description of the pattern to implement, to the pattern not being an effective one, or even due to a bug in the implementation.

Consider, for instance, the following rule which can be written by a fraud expert to identify taxpayers who declared low income for a number of years, which may be deemed to be suspicious:

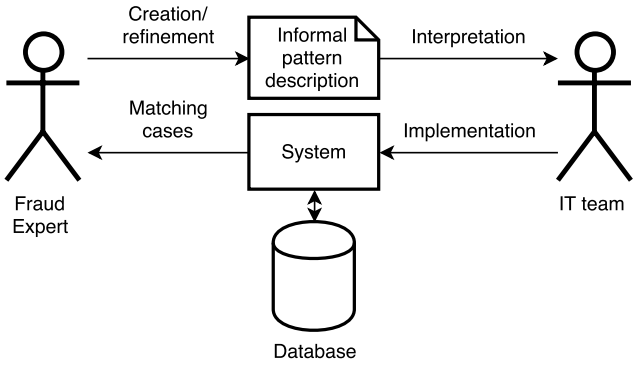

Fig. 1. Fraud detection process

Get taxpayer IDs for individuals who declared total income of less than 3000 Euro for any three years of assessment.

Some issues which may arise when applying the rule to extract reports from existing data are the following: Firstly, since the rule is expressed in plain English, it may be written imprecisely by the fraud expert, or be misinterpreted by the developer who might, for instance, interpret "total income" to refer to income from employment sources, and leave out other incomes such as pensions and bank interests. Furthermore, the manual development process is prone to coding errors, especially when dealing with multiple alterations of rules which require a number of changes.

Unexpected results due to these issues would require iterating through the process to ensure that the right reports are issued. However, these patterns identified by fraud experts are typically intelligent guesses, and it is desirable that they experiment with different rules, trying to identify ones which work better than others. Furthermore, constants such as the "3000 Euro" and the "three years" in this rule would be initial guesses intended to be refined interactively by the fraud expert. Would 3500 Euro be a better threshold in order to capture more potential fraud, or would 2500 Euro be better to get less candidate fraud cases thus affording more time to look at them more closely? For the fraud expert it is practically impossible to differentiate between the case when unexpected results are due to misinterpretation or a bug in the code, and when it is due to setting too high a threshold.

In this paper, we present an alternative approach to fraud detection, following the same methodology as currently in use, but by-passing the development process, thus reducing the points of failure. By having the fraud experts script their requests themselves, a correct compiler would allow them to focus directly on the development and refinement of fraud patterns. Since these experts are typically non-technical, the first challenge was to develop a controlled natural language (CNL) [Kuh14] for the domain of tax fraud rules, which allows them to write and execute rules directly. Another challenge is how to go from rule specifications written in the fraud CNL to executable code to process income information about entities and filter cases satisfying the rule. In order

to perform this, we have used runtime monitoring [LS09] techniques to compile CNL statements compositionally into stream processors which flag matching cases.

The approach has been implemented based on the actual data submitted in the Maltese Inland Revenue Department (IRD) system using the Grammatical Framework [AR10], and the tool and language were evaluated by involving tax fraud experts. Although the test population is small (due to the small number of tax fraud experts available), indications are that the level of abstraction of our language was appropriate to enable non-technical experts to understand and write rules and execute them to obtain fraud reports. The main contribution of the paper is the investigation of the application of a controlled natural language in a real-life setting, and the use of the language by non-technical experts to define runtime monitors.

The paper is organised as follows. In Section 2 we describe the framework we have developed. Section 3 describes the CNL we are proposing as well as its evaluation. The translation from rules written in the CNL to executable report generators is discussed in Section 4, in which we also examine the efficiency of the executable rules. We discuss similar work in Section 5, and finally conclude in Section 6.

2 A Fraud Monitoring Architecture

The main challenge we have addressed is that of empowering the non-technical fraud expert to write and experiment with fraud identification rules. Furthermore, the system enables the automatic extraction of information from a large database storing data about legal entities which is regularly updated with the submission of new tax documents. We have built a solution tailored to real-life data from the Maltese Inland Revenue Department (IRD) system. The challenge can be split into two sub-problems: (i) the design of an appropriate CNL to enable the writing of rules, and (ii) how such rules can be processed on a growing database of entries to check which taxpayers match particular rules.

The former problem has been addressed by developing a CNL focused on the actual needed concepts and grammar to address tax fraud patterns and which was discussed with a fraud expert working at the IRD during meetings held prior to the design of the system. The language has been implemented in GF [AR10] and allows non-technical users to input rules avoiding syntax errors. The latter problem, that of rule processing, is that the underlying rule execution engine has to embody the semantics of the CNL, but also ensure efficient processing of data — thus the semantics of the CNL would need to be interpretable in a serial manner, allowing for incremental analysis as new data and documents are received, requiring global re-evaluation only when new rules are set up.

With an ever changing dataset, another challenge for fraud experts is that of assessing the effectiveness of their new rules. We have adopted a tagged control dataset which can be used by the experts when experimenting with new rules. The limited size of the dataset ensures fast response, and the tagging (whether an entity is known to have been responsible for fraud) allows the system to report the percentage of false positives or negatives.

The resulting framework is shown in Fig. 2. This approach avoids going through a software developer every time a rule is added or modified, since the underlying concepts

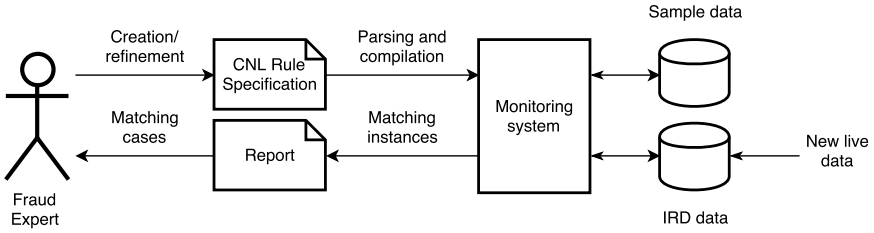

Fig. 2. Automated flow using a Controlled Natural Language

are hard coded into the system. The final result allows the fraud expert to refine the rules in a "what-if" manner where the feedback would indicate the accuracy of the rule. After analysing the rule on sample data, the fraud expert can target the full set of cases to be checked obtaining an accurate list of taxpayers which can be audited.

3 A CNL For The Specification Of Fraud Rules

The design of the CNL was crucial to ensure that fraud experts can identify fraud cases by querying a system using known patterns. These patterns involve the querying of tax submissions including patterns between different years of assessment. The language vocabulary and grammar were based on a corpus of queries identified through consultations with fraud experts from the Maltese IRD.

Apart from a basic ontology of terms related to financial and fraud concepts (such as employee, tax payer, income, expenses, tax credits and deductions) and relations between them (e.g. an employee is a tax payer, and an employee has an income), the language had to include (i) temporal notions, to be able to refer to time points and intervals in expressions (e.g. *"the income for the current year"* or *"the average income for the previous 3 years"*); and (ii) mathematical expressions in the form of arithmetic operators and numeric comparisons (e.g. *"income + 3000 Euro"*). Furthermore, the language had to include means of referring to aggregated values (e.g. *"the income for the current year is less than the income of each of the previous three years"*).

3.1 The Language

The fraud rule language uses a number of four basic concepts: (i) taxpayer filters, (ii) time-based sets, (iii) conditions, and (iv) reports. These are then combined at the top level to obtain full rules. The basic concepts are:

Taxpayer Filters. A key element of fraud rules is the choice of a subset of taxpayers on which a check is to be made. This includes categories of taxpayers, such as *employee*, *pensioner* and *company*, but also filters on data properties of the taxpayers as in the greyed out phrase in the following example:

Load the ID, where for any year, an employee of age more than 30 declared an income less than 3000 Euro.

These filters match the following grammar snippet:

$$
\begin{aligned}
\textit{Taxpayer} &::= \textit{taxpayer} \mid \textit{Individual} \mid \textit{Company} \mid \textit{Employer} \\
\textit{Individual} &::= \textit{IndividualType} \\
&\quad \mid \textit{IndividualType workAs Job} \\
&\quad \mid \textit{IndividualType age ComparisonOperator n} \\
\textit{IndividualType} &::= \textit{individual} \mid \textit{director} \mid \textit{employee} \\
\textit{Company} &::= \textit{CompanyType} \mid \textit{CompanyType operatingIn Industry} \mid \ldots \\
\textit{CompanyType} &::= \textit{company} \mid \textit{SME} \mid \textit{partnership} \mid \ldots
\end{aligned}
$$

Time-Based Sets. Another commonly occurring sentence component is that of temporal constraints and intervals. These are used in two distinct ways in the queries (i) specifying which years the rule is to be applied on e.g. "Load ID where for any 3 years ...", and (ii) give context to a field e.g. "... the income of each year is lower than the previous". The grammar has been defined to enable the identification of sets of years which are (i) sequential e.g. *any 5 sequential years* but also relative to a point in time as in *"the previous 4 years"*; (ii) open intervals of years e.g. *"the year 2009 onwards"*; (iii) arbitrary sets e.g. *"any 2 years"*; and (iv) singleton sets e.g. *"the current year"* and *"the year 2015"*. Furthermore, these can be combined as in *"for any 3 years from the year 2009 onwards"*, which intersects two year-set selectors.

Conditions. The conditions part of the language contains the checks to be met for a case to be flagged for auditing, typically using (i) a number of fields holding values such as *total income* and *profits*; (ii) aggregation operators such as *average*, *total* and *minimum*; and (iii) value comparison relations such as *less than*; (iv) trends on values or their change in availability such as *increase in income* and *stopped declaring profits*. These can be combined with year-sets to constrain aggregation operators e.g. *"the average income for years after 2009 is less than 2000 Euro"*.
In general, when the values throughout the rule refer to the current year of assessment, we give the option of leaving out the time reference e.g. *"income is less than expenses"* would be considered equivalent to *"income for the current year is less than expenses for the current year"*. In order to avoid ambiguity, we only allow this omission if there are no other references to years throughout the rule.
Finally, conditions can be combined using Boolean connectives as in:

Load the ID, where for any 3 sequential years from year 2009 onwards, an employee of age more than 30 declared a total income less than 3000 Euro or declared a decrease in employment income.

Reports. Another important element of the language is the ability to identify which fields are to appear in the fraud report using regular natural language construction:

Load the ID, age and total income for the last three years, where for any 3 sequential years from year 2009 onwards, an employee of age more than 30 declared a total income less than 3000 Euro or declared a decrease in employment income.

These language elements have been combined to obtain the fraud rule specification language, which takes a reporting clause, taxpayer identification clause, year-set constraint and filtering condition. Although the general structure of a rule is rather constrained, the freedom in the individual components allows for a wide range of rules covered by the language.

The language has been built within the Grammatical Framework (GF) [Ran11]. GF is a framework designed to address the translation of multiple languages by providing a number of CNLs in different languages. Furthermore, GF provides a number of morphological paradigms which aid in making the language more readable. GF was suitable since it has allowed us to create a custom-made language and apply these morphological inflictions on the language. The custom-made language was needed since the domain was too specific to use available generic CNLs.

The grammar contains around 160 rules, with approximately 1,000 lines of code. Most of the language has been built from scratch since there are too main domain specific concepts, but reusing morphological rules from the GF framework.

GF also offers a number of authoring tools, one of which is the Minibar[1] on-line editor. This has been used to display the grammar in a more suitable format. Using a predictive parser, the tool allows only valid entries, which was vital to make sure that all the rules are grammatically correct.

3.2 CNL Evaluation

The effectiveness of the fraud rule CNL we developed has been evaluated with six fraud experts working with the Maltese IRD and two accountants working in the private sector. Although the number of persons evaluating the system is low, it is worth noting that this includes practically all fraud experts working in the field with the IRD in Malta. Full details of the evaluation can be found in [Cal16].

The use of the language was evaluated during individual meetings, with each fraud expert being presented with the language in a one-hour session. The users were given a short demonstration of the language which was followed with four exercises assessing different aspects of how effective the language is: (i) *ability to read and understand the language*; (ii) *completeness of the terminology and concepts covered by the language*; (iii) *ability to write rules*.

For the first exercise, we took advantage of the fact that users were bilingual and were asked to explain a number of rules written using the CNL in Maltese, thus avoiding simple paraphrasing of the sentence. The second exercise consisted of two parts (a) they were asked to translate a number of fraud rules written in Maltese into English to assess whether the vocabulary and underlying concepts used were included in the CNL; and (b) they were asked to come up with three tax fraud rules and write them in natural language to assess whether the grammatical constructs used were allowed in the CNL. For the final test, the users were asked to pick one of the rules identified in the previous exercise and write it using the authoring tool in our CNL.

[1] http://cloud.grammaticalframework.org/gfse/

The users managed to understand almost all the rules presented to them, with only one rule which was not fully understood by two fraud experts. The rule used two instances of year selections which can lead to potential ambiguity in interpretation:

> *Load ID where for any 3 years, an individual declared average total chargeable income for the previous 3 years less than 2000 Euro.*

Individual differences in preferences with regards to the language were minimal, and could easily be catered for by extending the vocabulary, for instance, to include the word *times* as an alternative to the mathematical symbol $*$ as preferred by one user.

When asked to identify fraud rules, and to implement one of them using the CNL, all users identified rules which were covered by the language, and were correctly expressed. However, there were cases were fraud experts wanted to refer to fields which were not available in the proof-of-concept tool developed. For the final version of the tool, this would have to be extended to the full set of fields available from the IRD database — something which can be done in a straightforward manner.

The evaluators were also asked questions regarding the use of such a language to query information as opposed to using spreadsheets (as they usually do). Only one user preferred using a spreadsheet, but adding that this preference came from years of being accustomed to using spreadsheets due to an accounting background. It was clear that while all the users became familiar with the language during the one-hour evaluation session, a longer session would be necessary in order to cover further detail and, for instance, discuss which possible rules can be written, while also giving the users more information regarding the tool. This is needed since the language can become complex when using time-based logic.

4 Monitoring Fraud Rules

One way of giving an executable semantics to the Tax Fraud CNL is to see the database of taxpayers' information as a large repository of time-stamped data, and the rules corresponding to database queries. However, such an approach requires that it is rerun whenever new data is added to the database, and when one considers that the IRD database can be rather large, this can result in hours of processing for each complex set of rules. Since the data typically arrives in temporal order, however, one can adopt a more incremental approach, processing the data as it comes in, and keeping a state to avoid recalculating things repeatedly. To contrast the approaches, consider a rule which states:

> *Load the ID, age and income for the current year, where for any three sequential years, a taxpayer declared an income less than 3000 Euro.*

Whenever new information about some taxpayers appears, the former global approach queries the database for data from any three sequential years about a taxpayer, and processes the rest of the logic on the data collected. Unless concrete logic is added to store the fact that for some year intervals this analysis has already been done, earlier years will be reprocessed whenever new data becomes available. In contrast, if we use an incremental approach which stores whether each taxpayer has declared less than

3000 Euro in these past two years, analysis on new data can be performed efficiently by (i) checking that the threshold has not been exceeded with the data from the new year and report if the rule is satisfied, and (ii) update the state appropriately.

There has been much work in runtime monitoring [LS09] on building techniques to address such situations efficiently. It was thus decided that we adopt a standard runtime verification tool, Larva [CPS09], to process the data efficiently, using techniques from [CP13,CPA09]. Larva allows for specifications to be written in a guarded command language format, possibly structured using automata — although for the sake of encoding the semantics of our CNL, the guarded command rules sufficed.

Rules take the form of: *event* | *condition* $\mapsto$ *action*, indicating that whenever the event (document being submitted, data becoming available, etc.) happens, and the condition is satisfied, the action is executed. For instance, consider a rule which states that *"*[*some condition holds*] *for three consecutive years"*. This can be encoded by introducing a variable *count* which keeps count of the number of past consecutive years (up to 3) in which the condition held. The count is initialised to zero, and the three following rules (i) increase the count when the condition is satisfied; (ii) resets the count when it is not; and (iii) reports a match when the count has reached 3[2] with the greyed out parts to be replaced by appropriate code depending on the rest of the rule:

$$\begin{array}{l} \textit{some condition holds} \mid \textit{count} < 3 \mapsto \textit{count}++ \\ \textit{some condition holds} \mid \textit{count} = 3 \mapsto \textit{matches} \\ \neg\ \textit{some condition holds} \mid \textit{true} \mapsto \textit{count} = 0 \end{array}$$

Furthermore, Larva allows for replication of rules for different instances of the same object, thus allowing us to structure the rules above to be run for each possible taxpayer:

$$\begin{array}{l} \textit{foreach } p : \textit{TaxPayer} \\ \quad \textit{some condition holds} \mid p.\textit{count} < 3 \mapsto p.\textit{count}++ \\ \quad \textit{some condition holds} \mid p.\textit{count} = 3 \mapsto \textit{matches} \\ \quad \neg\ \textit{some condition holds} \mid \textit{true} \mapsto p.\textit{count} = 0 \end{array}$$

In order to take appropriate action depending whether an instance matches, Larva allows for communication channels to be used between rules. Whenever a match occurs, we can send a message with the taxpayer's information on a particular channel: $\textit{matches}(p)!$, which is consumed by a reporting rule. For instance, a rule starting *"Load the ID, age and income for the current year…"*, would be encoded as the follows:

$$\textit{matches}(p)? \mid \textit{true} \mapsto \textit{load}(p.\textit{ID}, p.\textit{Age}, p.\textit{income}(\textit{currentYear}))$$

In order to implement time-based checks such as *"average income for the previous three years is less than 3000 Euro"*, the system stores information from one year to another. In order to avoid storing all the available data, the conditions need to address two aspects: (i) the respective condition to be implemented as a rule, and (ii) to store the yearly information of the field in question. These two aspects are addressed in different

[2] The semantics of the rules is such that the conditions are all checked before any actions have taken place, thus avoiding race conditions.

sets of rules, therefore the structure available in GF was crucial for this to be possible. By using a template-based approach to generate Larva code, with generic solutions such as monitors to keep track of counters and frequencies, and which are specialised to deal with the objects referred to in the CNL specification.

Using GF, we have encoded these translations in a compositional manner on the grammatical structure of the rules, encoding the monitors as a different language into which the rules are translated. Clearly, for the monitoring language, GF support for morphological inflections was not required, keeping the number of language structures in Larva smaller.

In the performance evaluation, our approach was deemed to perform well, and in fact, a sample of 53,000 records were checked in less than four seconds. This ensures that the fraud expert is given the report in a timely manner which is one of our aims. Furthermore, checking 3.2 million records, the system took around 3 minutes and this meant that rules can be executed on large sets of data and still retrieve feedback in a reasonable amount of time. When comparing these performance measures to traditional database queries, we found that unless carefully optimised, such queries are significantly less efficient as they would have to be reapplied globally every time new data becomes available. Optimising such queries involves non-straightforward manipulation of the querying code to introduce indexes and tables with intermediate results, thus potentially being error prone and as such undesirable in our context.

5 Related Work

Jones et al. [JES00] have shown that with careful choice of syntax, even low-level (as opposed to natural) domain-specific languages can achieve a high-level of readability by non-technical experts — in their case, they present a combinator library to construct financial contracts defined by financial engineers. In contrast, despite the end-users' similar backgrounds in finance, in our initial meetings with fraud experts, the use of such low-level notation was frowned upon, which led us to go for a more natural, albeit controlled, syntax.

For similar reasons, we avoided the use of a template-based approach e.g. [PSE98], in order to allow for a more granular and compositional grammar. The approach used in our work combines the grammar-based and template-based approaches by presenting a high-level template to the users to make it more understandable, whilst still using a pool of underlying core concepts.

The use of controlled natural languages as a means of providing a flexible input format for non-technical experts to be able to express instructions is rather widespread e.g. [CGP15]. The natural aspect of the language, especially if used with an appropriate user interface supporting syntactically correct-by-design input, allows for end-user development within specific domains. GF itself has been used for various such case studies e.g. [DRE13,RED10]. What distinguishes our approach from most other similar work is the semantic interpretation of the language, and the use of a runtime monitoring approach to allow for stream-based data analysis derived from natural language queries. For instance, Dannéls et al. [DRE13], build a CNL tackling museum system queries. As in our case, theirs is focused on a particular domain, that of identifying paintings in a

system. In contrast, however, our language is more controlled and technical and less natural than theirs, which was required to be able to give a semantic interpretation of the terms into a stream processing monitor.

We have already explored the use of monitoring techniques as a backend for a CNL in [CGP15]. However, the fraud language we present in this paper is substantially more extensive and expressive, even if backend techniques are similar.

6 Conclusions

In this paper, we have presented a controlled-natural-language-driven framework aimed at supporting fraud experts to be able to, in an autonomous manner, explore different fraud rules and apply them to a live set of taxpayer data. The backend of the framework has been developed as an incremental monitor, enabling sufficiently efficient analysis of large datasets — experiments show that running a number of rules on the data from 53,000 documents takes less than four seconds, an improvement on simple database queries. This limited dataset, however, is used just to enable fraud experts to assess their rules, before running them on the whole dataset with over 3.2 million documents and which were processed in approximately three minutes.

The CNL we have developed has been evaluated by a number of fraud experts currently working on real-life tax fraud detection, which showed that the language was effective both in enabling them to write rules, but also to correctly understand and interpret rules written by others. The language currently contains the basic concepts of numeric and monetary values in order to enable fraud experts to use it, however, we plan to extend the language to (i) the full set of fields found in the taxation documents; and (ii) include richer operations e.g. extend the predicate "increase in income" to be able to access a finer grained "percentage increase in income".

One interesting aspect in our use of GF is the multi-lingual support it provides which can be harnessed to present the CNL in multiple languages while keeping the same monitoring system. Since many taxation concepts are common across countries, we envisage that this should be feasible with only minor localisation issues. Although we have not addressed this in our work, we have instead treated our compilation into monitors as a GF translation. Although our approach is very constrained, it might be worth investigating further the use of translation support for compilation into executable code as a means of semantic analysis or inspection.

References

[AR10] Krasimir Angelov and Aarne Ranta. Implementing controlled languages in gf. In *Proceedings of the 2009 Conference on Controlled Natural Language*, CNL'09, pages 82–101, Berlin, Heidelberg, 2010. Springer-Verlag.

[Cal16] Aaron Calafato. A domain specific property language for fraud detection to support agile specification development. Master's thesis, University of Malta, 2016.

[CGP15] Christian Colombo, Jean-Paul Grech, and Gordon J. Pace. A controlled natural language for business intelligence monitoring. In Chris Biemann, Siegfried Handschuh,

Andre Freitas, Farid Meziane, and Elisabeth Metais, editors, *Natural Language Processing and Information Systems*, volume 9103 of *Lecture Notes in Computer Science*, pages 300–306. Springer International Publishing, 2015.

[CP13] Christian Colombo and Gordon J. Pace. Fast-forward runtime monitoring an industrial case study. In Shaz Qadeer and Serdar Tasiran, editors, *Runtime Verification*, volume 7687 of *Lecture Notes in Computer Science*, pages 214–228. Springer Berlin Heidelberg, 2013.

[CPA09] Christian Colombo, Gordon J. Pace, and Patrick Abela. Offline runtime verification with real-time properties: A case study. In *Proceedings of WICT 2009*, 2009.

[CPS09] Christian Colombo, Gordon J. Pace, and Gerardo Schneider. Larva — safer monitoring of real-time java programs (tool paper). In *Seventh IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, pages 33–37. IEEE Computer Society, November 2009.

[DRE13] Dana Dannélls, Aarne Ranta, and Ramona Enache. Multilingual grammar for museum object descriptions. *Frontiers of Multilingual Grammar Development*, page 99, 2013.

[JES00] Simon P. Jones, Jean M. Eber, and Julian Seward. Composing contracts: an adventure in financial engineering (functional pearl). In *ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 280–292, New York, NY, USA, 2000. ACM.

[Kuh14] Tobias Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, March 2014.

[LS09] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[MTVM93] Sam Maes, Karl Tuyls, Bram Vanschoenwinkel, and Bernard Manderick. Credit card fraud detection using bayesian and neural networks. In *In: Maciunas RJ, editor. Interactive image-guided neurosurgery. American Association Neurological Surgeons*, pages 261–270, 1993.

[PSE98] Richard Power, Donia Scott, and Roger Evans. What you see is what you meant: direct knowledge editing with natural language feedback. In *ECAI*, pages 677–681, 1998.

[Ran11] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. Center for the Study of Language and Information/SRI, 2011.

[RED10] Aarne Ranta, Ramona Enache, and Gregoire Detrez. Controlled language for everyday use: The molto phrasebook. In Michael Rosner and Norbert E. Fuchs, editors, *CNL*, volume 7175 of *Lecture Notes in Computer Science*, pages 115–136. Springer, 2010.

[TD99] David M. J. Tax and Robert P. W. Duin. Data domain description using support vectors. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 251–256, 1999.