SPECIFYING AND COMPILING HIGH LEVEL FINANCIAL FRAUD POLICIES INTO STREAMSQL

Michael Edward Edge, Pedro R. Falcone Sampaio
Manchester Business School, University of Manchester, Booth Street East, Manchester, M16 6PB, U.K.

Oliver Philpott
School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K.

Mohammad Choudhary
Sparta Technologies Ltd, Northern Incubation Unit, Sackville Street, Manchester, M60 1QD, U.K.

Keywords: Fraud Management, Internet Fraud, Policy-Based Languages, Data Stream Processors, Compilers.

Abstract: Fraud detection within financial platforms remains a challenging area in which criminals continue to thrive, breaching security mechanisms with increasingly innovative and sophisticated system attacks. Following the migration from reactive to proactive screening of transactional data to reduce an organisations fraud detection latency, fraud analysts now find themselves responsible for the maintenance of extensive fraud policy sets and their implementation as complex data stream processing procedures. This paper presents a Financial Fraud Modelling Language and policy mapping tool for high level expression and implementation of proactive fraud policies using stream processors. A key aspect of the approach is reduction of the complexity and implementation latency associated with proactive fraud policy management through abstraction of policy functionality using a conceptual level modelling language and innovative policy mapping tool. This paper focuses upon the rule based language model for high level expression of financial fraud policies and the associated compiler tool for specifying and mapping policies into StreamSQL.

1 INTRODUCTION

Banking institutions have a strong interest in increasing the speed at which fraudulent activity can be detected due to its direct relation to financial loss, customer service and the organisations status as reputable financial provider. Existing research into fraud detection mechanisms based upon data mining has a limited capability to address the expanding number of ubiquitous electronic delivery channels for financial services due to the alerting latency incurred through post transactional analysis over a finite data store (reactive fraud management) (Kou, Lu et al. 2004; Phua, Lee et al. 2005). Accordingly, emerging technologies are employing real-time processing models for continuous monitoring of streaming service channel data and triggering of a preventive response prior to transaction completion, minimising the potential fraud deficit (proactive fraud management) (Entrust 2008; Fair Isaac 2008; StreamBase 2008). Despite the successful shift of fraud analytics from 'post' to 'pre' data storage, many current proactive solutions are capable of fraud management only upon a single channel, with no support for fraud analysis over multiple incoming data delivery channels. More crucially, few systematic methods exist for assisting fraud analysts in the modelling and enforcement of anti-fraud policies using stream processors, with many solutions based upon code implementation for low level stream application programming interfaces (Chandrasekaran 2003; Arasu, Babu et al. 2006) and other low level imperative event languages (Luckham 2005).

This paper outlines the development of a Financial Fraud Modelling Language (FFML) and policy mapping architecture for the conceptual level modelling and implementation of fraud policies using StreamSQL, an emerging standard for

Edward Edge M., R. Falcone Sampaio P., Philpott O. and Choudhary M. (2009).
SPECIFYING AND COMPILING HIGH LEVEL FINANCIAL FRAUD POLICIES INTO STREAMSQL.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 194-199
DOI: 10.5220/0002001501940199

processing real-time data streams. Specifically it details the development and implementation of a compiler component for the automated parsing of FFML policy definitions and generation of the required stream processing representation. A key element of the work is abstraction of low level stream processing syntax through conceptual level Event-Condition-Action policies to reduce the complexity and implementation latency associated with proactive fraud policy enforcement.

The remainder of this paper is organised as follows: Section 2 presents a background on proactive fraud management and the FFML policy management framework. Section 3 describes the FFML policy definition language. Section 4 presents the design and implementation of the FFML compiler tool. Section 5 illustrates a sample FFML to StreamSQL mapping. Section 6 details the key contributions of the work. Section 7 presents a summary of the work and outlines future research.

2 BACKGROUND

Financial Fraud Modelling Language (FFML) provides a conceptual level modelling and fraud prevention architecture using a rule based modelling syntax to assist fraud analysts in the expression and assembly of proactive fraud policy sets prior to representation within target stream processing solutions (Edge, Sampaio et al. 2007).

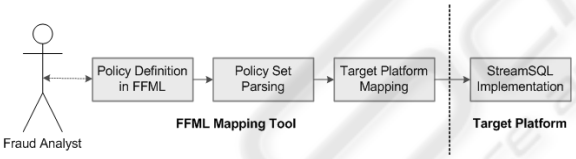

Figure 1: FFML Policy Mapping Approach.

Figure 1 illustrates the FFML policy mapping approach for translation of fraud policies into the required stream processing syntax. Fraud policy set definition is undertaken through a front end GUI component for the conceptual level management of complete fraud policy sets using FFML. Automated parsing validates assembled policy sets against the FFML language specification to ensure defined policies are well formed from which the corresponding stream processing syntax is generated using the required target platform code generation component. Target platform adaptors are implemented in a plug and play architecture facilitating the mapping of FFML policies into multiple stream processing implementations, leveraging a dynamic and extensible policy management architecture.

3 FRAUD POLICY SPECIFICATION USING FFML

FFML provides a domain specific language of constructs and operators to facilitate the expression of rule-based financial fraud policies which may span multiple service channels, time windows and transaction event types, without the restrictions and extensive programming requirements of the employed target platform. FFML policies are assembled as Event-Condition-Action (ECA) definitions using a domain specific language of constructs and operators to support the conceptual level expression and management of proactive fraud policy controls (Table 1).

Table 1: FFML Policy Structure.

Operator	Parameter
POLICYID	policy reference
ON	event statement
IF	condition statement
THEN	action statement

3.1 Event Statement

Policy event triggers define the click stream data patterns to which policies continually monitor incoming transaction service channels for invocation of defined evaluative functionality. Table 2 illustrates the expressiveness of the FFML event model using the following sample fraud policy definition: *"if there is an online banking session containing a password change event followed by funds transfer, or a session containing a failed logon phase 1 attempt, a failed logon phase 2 attempt and a funds transfer, require two factor authentication for transaction completion"*. Channel selectors are first declared specifying the incoming data stream to which the policy applies, followed by the event, or event sequence upon which evaluation of defined conditions should be performed. Event sequences support the matching of chronological data stream events using the FFML *"SEQ"* operator for specifying time window durations during which defined behaviour will be matched following occurrence of the initial sequence event. The policy definition in Table 2 illustrates the use of 5 minute time windows (300 seconds) for matching of the specified online user behaviour.

Table 2: Simple Online Fraud Policy.

Online Fraud Policy	
POLICYID	ONL01234
ON	ONL **SEQ**(300)[passwordchange, transfer] **OR** ONL **SEQ**(300) [failed_logonphase1, failed_ logonphase2, transfer]
THEN	TWOFACTOR(transid, sortcode, accountnumber);

In the specification of event statements and sequences, the following principle is applied: the occurrence of particular events implicitly assumes the occurrence of any prerequisite events. This achieves syntax reductions by eliminating the need for definition of events which maybe implicitly assumed based upon preceding sequence events. For example, in Table 2 "*passwordchange*" implicitly assumes the user has completed a successful logon process. Similarly, definition of the "*transfer*" event following "*failed_logonphase2*" also assumes that a successful logon has taken place between these two event occurrences. Table 2 also illustrates how multiple event trigger specification is supported using the disjunctive "*OR*" operator providing that a common final channel event exists, eliminating the need for multiple policy definitions which require the same evaluative functionality.

Conjunction between event instances has been restricted due to the window based processing model of underlying stream processing technologies. Matching of event/event sequences between user transaction sessions would open extensive time windows spanning several hours, or even days for matching of specified event instances, which is clearly unfeasible in a global user service and would have significant performance ramifications within the supporting business platform. For example, Table 3 specifies the following sample fraud policy definition: "*if there is over £500 of Card Not Present transactions in the last 24 hours, followed by an online banking session containing a failed logon phase 1 attempt, a failed logon phase 2 attempt and a transfer with value greater than or equal to £1500, trigger an alert"*.

Table 3: FFML Online Policy Definition.

Online Fraud Policy Definition	
POLICYID	ONL01234
ON	ONL **SEQ**(300)[failed_logonphase1, failed_logonphase2, transfer]
IF	**QUERY** TOTALDEBIT(CNP, sortcode, accountnumber) >= 500 **AND** value >= 1500
THEN	ALERT(transid, sortcode, accountnumber);

Preceding transactional account behaviour is traced through implementation of stored data operators within the defined condition, using the latter policy event for triggering of condition evaluation. Accordingly, policy evaluation is only performed upon matching of the specified transfer transaction, rather than opening of extensive time windows following each CNP transaction for detection of the subsequent online event sequence. Table 3 illustrates the use of the 'TOTALDEBIT' query for retrieval of all CNP transactions within the last 24 hour period.

Table 3 also illustrates how the developed modelling language supports multi-channel risk models through policy triggering in response to events within one streaming data channel, while supporting condition evaluation over account transactions performed through other service channel provisions. FFML therefore facilitates definition of sophisticated policies which encompass fraud evaluation over all delivery service channels towards the deployment of increasingly integrated and holistic fraud detection frameworks.

3.2 Condition Statement

Condition functionality is defined as a series of Boolean logic statements assembled using disjunctive (*"OR"*) and conjunctive *("AND")* operators for evaluation of system transactions which satisfy defined event triggers using incoming event data, stored data functions and arithmetic operators.

A key element of the FFML language is the ability to define policies which evaluate streaming values against post-transactional data from both current and past financial trading (Table 4), enabling a fraud decision to be made based upon examination of preceding account behaviour rather than through isolated queries on streaming transactional data alone. Table 5 illustrates how the developed approach maybe used to express the following sample fraud policy scenario: "*If there is an ATM withdrawal that reaches the total daily £250 withdrawal limit, and the daily limit has been reached 3 times within the last 5 days, stop the transaction, raise alert and block account"*.

Data parameters utilised within Boolean functions and stored data operations are associated with the last declared event in each event trigger to achieve syntax reductions within condition specification. While this principle restricts the use of the parameters associated with preceding non-transactional event instances, it is emphasised that such instances are used only to assist in the identification of suspicious transactions, and alone are unbeneficial for fraud policy definition.

Accordingly, event sequence specifications shall always feature one or more non-transactional events, followed by a concluding transactional instance for triggering of condition evaluation.

Table 4: Stored Data Retrieval Operators.

Function	Database Query
Syntax	**QUERY**
Parameters	QUERY_NAME(parameters) || (SELECT….FROM….WHERE)
Description	Issue pre-defined queries against stored transactional data for the current financial day or cumulatively over multiple financial days if a day parameter is provided. Standard SQL maybe utilised for custom data evaluation functions. Result must return a single value to be used within a Boolean condition. (Note: Day parameters not compatible with HISTORY function – see below.)
Function	History Operator
Syntax	**HISTORY**
Parameters	(days)[condition]
Description	Evaluates complete conditions over multiple financial days returning an integer indicating the number of days on which the query evaluated to true. Condition contains one or more QUERY components. HISTORY functions are therefore use for issuing complete conditions over each financial day, while QUERY operators simply return a cumulative figure for the specified preceding period.

Table 5: ATM Fraud Policy.

ATM Fraud Policy	
ON	ATM[withdrawal]
IF	**QUERY** TOTALDEBIT(ATM, sortcode, accountnumber) + value >= 250.00 **AND** **HISTORY**(4) [**QUERY** TOTALDEBIT(ATM, sortcode, accountnumber) >= 250.00] >=2
THEN	BLOCK(sortcode, accountnumber) **AND** ALERT(transid,sortcode, accountnumber);

3.3 Action Statement

Actions specify the preventive response to be triggered within the supporting business platform upon successful triggering and evaluation of the defined policy instance. FFML actions are categorised into two distinct categories; *active* and *passive*. Passive actions are regarded as those actions which do not alter the current transaction path and enable the transaction to complete as normal, for example if an *'ALERT'* or *'FLAG'* is applied for post-transaction examination of the account by the fraud analyst. Active actions are those which cause transactions to deviate from their normal execution path, for example if a 'BLOCK' request is issued against a particular account or two factor authentication is initiated for confirming the identity of the initiating account holder. All active actions are implicitly assumed to apply the "DOINSTEAD" principle for transaction execution, as described in (Stonebraker, Jhingran et al. 1990). Multiple actions are applied using the conjunction operator which are mapped onto the required output stream for examination by fraud personnel and triggering of the specified preventive operations.

4 FFML GUI AND POLICY COMPILATION TOOL

The FFML tool provides fraud analysts with a suite of tools for the construction, manipulation and compilation of FFML policy sets. The developed graphical front-end component comprises a source code editor, parser, compiler and file management system, for rapid policy set construction and management of an organisations fraud policy deployment from a single point of control (Figure 2).

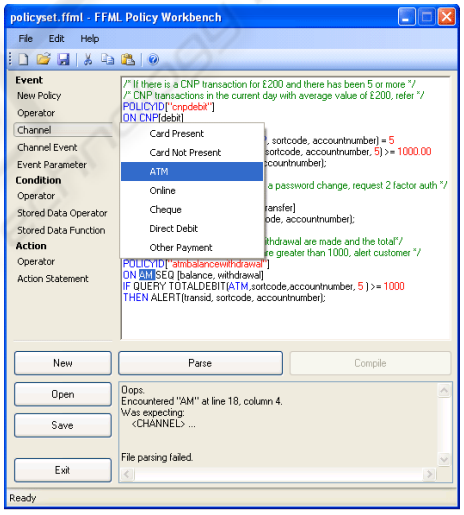

Figure 2: FFML GUI Tool.

4.1 Compiler Architecture

FFML policy statements are translated into the target syntax model using an automated compiler component, implemented using the JavaCC parser generator. More specifically, the compiler is specified using JJTree (a pre-processor for JavaCC), enabling the insertion of tree-building actions into the JavaCC grammar for generation of Abstract Syntax Tree (AST) definitions utilised in the validation of FFML policies and generation of target implementation code. Figure 3 demonstrates how the developed components are used for validation

and mapping of FFML statements into the target syntax model.

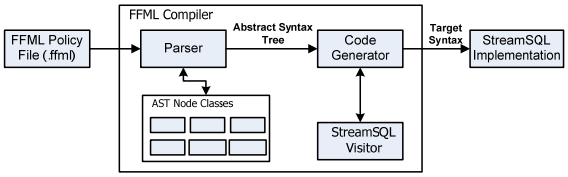

Figure 3: FFML Compiler Architecture.

4.2 Code Generation

Stream processing implementations such as StreamSQL implement computational functionality through the assembly of data functions and internal streams within a data processing network. A clear requirement for mapping of policies onto stream processing language models is the need to maintain a state between node visits to facilitate the exchange of internal stream references between generated data functions. The utilised visitor design pattern addresses this requirement through the logical grouping of visit methods using a "syntax separate from interpretation" programming model for all syntax tree nodes, facilitating the use of collections and other programming data structures within each interface implementation for internal reference storage.

The visitor design pattern also supports the mapping from conceptual level policy definitions into multiple target language implementations without extensive re-engineering of the supporting compiler component. New target language implementations are supported through development and integration of the necessary visitor module for expressing the mappings from FFML grammar productions to the corresponding stream based operations. Run-time binding of target platform adaptors therefore creates a highly dynamic and extensible architecture for fraud policy management in fragmented and multiple disparate fraud management environments.

5 POLICY MAPPING EXAMPLE

Table 3 (Section 3.1) presents a sample FFML policy definition for an online banking channel. Translation of the policy statement requires construction of several StreamSQL target operators and associated internal streams for the feeding of information through the data processing network. Appendix A illustrates how implementation of the sample fraud policy scenario therefore requires a total of 52 lines of StreamSQL code. Table 3 expresses the same policy functionality using just 6 lines of FFML, resulting in over an 80% reduction in the required syntax compared to direct implementation within the target stream processing model. This is seen as a significant advancement for assisting fraud analysts in the definition and maintenance of proactive fraud policy controls using stream processors.

6 KEY CONTRIBUTIONS

Conceptual Level Specification of Proactive Fraud Policies - Poor declarative specification of complex policies often results in unorganised distribution of policy code throughout target platform implementation code, which requires significant re-engineering in subsequent development phases at a substantially escalated cost. FFML provides a domain specific language for the expression and management of proactive fraud policies over multiple streaming channels and differing time windows from a single modelling perspective. Complexity is therefore abstracted from low level implementation of fraud controls to enable conceptual level construction of complete fraud policy sets usable by both expert and non-expert users.

Automated Policy Set Implementation in a Stream Based Language – Plug and play target platform adaptors encapsulate the semantic knowledge for mapping of FFML policies to simplify the complexities associated with direct implementation of large scale policy sets within explicit low level programming formalisms. FFML therefore provides an innovative policy management architecture supporting the mapping of policies to multiple disparate target implementations and future changes to underpinning fraud technologies through simple re-mapping of an organisations fraud policy set to new target syntax models.

Improved Responsiveness and Policy Set Realignment– FFML significantly enhances an organisations responsiveness to fraud threats through reduction of the implementation latency associated with future maintenance operations to existing fraud policy sets. New policies and modifications to existing policies may be rapidly implemented through the developed GUI environment using the FFML toolset for abstraction of the complexity associated with management of policies directly within low level target language models. Furthermore, expression of fraud policies within a common conceptual level language supports the

active sharing of fraud policy data between financial sector organisations using a service oriented architecture, significantly reducing the latency associated with discovery and deployment of fraud policies in response to emerging industry threats (Edge, Sampaio et al. 2008).

7 SUMMARY

This paper presents a financial fraud policy specification language and policy mapping technology for simplifying the challenges associated with proactive fraud policy management using stream processors. Fraud policies are defined using a domain specific modelling language (FFML) and translated into a StreamSQL representation using the developed compiler component. A key element of the framework is the application of an Event-Condition-Action model for specification of proactive fraud policies which span multiple channels, time windows and events, and mapping into the required stream processing implementation. It is also illustrated using a simple example how the expression of fraud policies using FFML can result in significant syntax reductions over direct implementation within the underlying stream processing language model. Future work will include the development of a real-time customer profiling mechanism using signature-based models (Edge and Sampaio 2009) and a component for optimisation of generated StreamSQL code.

REFERENCES

Arasu, A., S. Babu, et al. (2006). "The CQL continuous query language: semantic foundations and query execution." The VLDB Journal 15(2): 121-142.

Chandrasekaran, S. (2003). TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR

Edge, M. E. and P. R. F. Sampaio (2009). "A Survey of Signature Based Methods for Fraud Detection." Computers and Security [To appear].

Edge, M. E., P. R. F. Sampaio, et al. (2007). Towards a Proactive Fraud Management Framework for Financial Data Streams. The 3rd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC'07), Loyola College Graduate Center, Columbia, MD, USA., IEEE.

Edge, M. E., P. R. F. Sampaio, et al. (2008). A Policy Distribution Service for Proactive Fraud Management over Financial Data Streams. IEEE International Conference on Services Computing, 2008. (SCC '08), Honolulu, Hawaii, USA.

Entrust. (2008). "www.entrust.com."

Fair Isaac. (2008). "www.fairisaac.com."

Kou, Y., C.-T. Lu, et al. (2004). Survey of fraud detection techniques. IEEE International Conference on Networking, Sensing and Control.

Luckham, D. (2005). The RAPIDE Pattern Language. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Pearson Education: 145 - 174.

Phua, C., V. Lee, et al. (2005). "A Comprehensive Survey of Data Mining-based Fraud Detection Research."

Stonebraker, M., A. Jhingran, et al. (1990). "On Rules, Procedures, Caching and Views in Data Base Systems." Proceedings of the 1990 ACM SIGMOD: 281 - 290.

StreamBase. (2008). "www.streambase.co.uk."

APPENDIX

```
CREATE INPUT STREAM ONL_failed_logonphase1(
transid string(10), sortcode string(6),
accountnumber string(8), datetime timestamp,
onlineid string(20), ipnumb string(25),
sessionid string(25), password1_entered
string(25));

CREATE INPUT STREAM ONL_failed_logonphase2(
transid string(10), sortcode string(6),
accountnumber string(8), datetime timestamp,
onlineid string(20), ipnumb string(25),
sessionid string(25), password2_entered
string(25), password3_entered string(25));

CREATE INPUT STREAM ONL_transfer(
transid string(10), sortcode string(6),
accountnumber string(8), datetime timestamp,
onlineid string(20), ipnumb string(25),
sessionid string(25), currency string(3),
amount double, dest_sortcode string(6),
dest_accountnumber string(8), dest_transferdate
string(10));

CREATE  STREAM out__Pattern_1;
SELECT ONL_transfer.transid AS transid,
ONL_transfer.sortcode AS sortcode,
ONL_transfer.accountnumber AS accountnumber,
ONL_transfer.onlineid AS onlineid,
ONL_transfer.ipnumb AS ipnumber,
ONL_transfer.sessionid AS sessionID,
ONL_transfer.currency AS currency,
ONL_transfer.amount AS amount,
ONL_transfer.dest_sortcode AS dest_sortcode,
ONL_transfer.dest_accountnumber AS
dest_accountnumber,
ONL_transfer.dest_transferdate AS
dest_transferdate
FROM PATTERN ((ONL_failed_logonphase1 THEN
ONL_failed_logonphase2)
THEN ONL_transfer) WITHIN 300 TIME
WHERE  ONL_failed_logonphase1.transid  =
ONL_failed_logonphase2.transid
AND ONL_failed_logonphase2.transid  =
ONL_transfer.transid INTO out__Pattern_1;

CREATE STREAM out__TOTALDEBIT_2;
APPLY JDBC accountdata
"SELECT sum(amount) AS currentdaytotal FROM
transactions
WHERE (channel = 'CNP')
AND sortcode = {sortcode} AND accountnumber =
{accountnumber}
AND type = 'deb'
AND transdate >=
CONVERT(datetime,(FLOOR(CONVERT(float(GETDATE()
)))
AND transdate <
CONVERT(datetime,FLOOR(CONVERT(float,DATEADD(dd
,1,CURRENT_TIMESTAMP))));" FROM out__Pattern_1
INTO out__TOTALDEBIT_2;

CREATE STREAM out__Filter_3;
SELECT * FROM out__TOTALDEBIT_2
WHERE currentdaytotal >= 500 INTO
out__Filter_3;

CREATE STREAM out__Filter_4;
SELECT * FROM out__Filter_3
WHERE value >= 1500 INTO out__Filter_4;

CREATE OUTPUT STREAM ALERT;
SELECT transid AS transid,
sortcode AS sortcode ,
accountnumber AS accountnumber
FROM out__Filter_4 INTO ALERT;
```