1 Fraud Detection and Business Process Module

This section describes the design of the Artificial Intelligence (AI) module supporting the Fraud Detection (FD) and Business Process (BP) aspects of network management provided by the MDS. The content is outlined below, and covers design processes, data analysis and prototype results, as well as describing software engineering concerns.

1.1 Background

The tasks of the fraud detection (FD) and business process (BP) module are defined in the Description of Work as:

- FD — "finding fraudulent uses by subscribers"
- BP — "the pattern of uses generated by clients"

The source data for FD/BP comprises sets of Call Detail Records (CDRs), stored by the billing system and describing individual call events. CDRs may include a range of information, such as:

- Phone number
- Other party phone number
- Direction (to network, from network, forwarded)
- Call start date
- Completion status (completed, dropped)
- IPS user label (base station)
- Call duration

An individual call does not carry much information about subscriber behaviour, so it is necessary to build a profile of behavioural patterns over a longer period, encompassing, possibly, many calls. Such profiles serve two purposes: they allow deviations from established norms to be detected, and they facilitate the characterisation of subscribers by behavioural patterns. The processes of profiling CDRs and identifying variations in patterns of behaviour are identical, regardless of whether those variations are related to FD or BP, hence, these aspects are treated together. It is the domain expert's task to determine the impact, or otherwise, of the results from the FD/BP AI module of the MDS.

The purpose of the MDS is to explore the capabilities of selected Artificial Intelligence (AI) techniques for addressing various network management challenges, and to offer a proof-of-concept implementation. AI systems typically require some form of training process prior to deployment (although incremental learning / online adaptation is possible). Hence, an important focus of the design phase is a detailed consideration of the significant learning paradigms and their implications. Indeed, from a software engineering perspective, concerns focus more on the API and cross-boundary interactions, rather than low level design and algorithmic details.

The following sections outline the process involved in the design of the AI module and explore the AI context, before progressing to describe the data analysis and prototypes, and, then, the system context and module design, in sufficient detail to facilitate understanding of its function and interaction with other parts of the MDS. Finally, quality of service requirements are considered, with proposals for the assessment of performance during the evaluation phase of the project.

1.2 Process

This section of the design document reports upon the current state of development of the FD/BP AI module. In this sub-section, the processes involved in, and leading up to, the design of said module, are outlined. The main inputs to the design process came from the following sources:

a. *Review of the literature concerning AI techniques for MDS.* A comprehensive review was developed, and reported in D2.2.1. The purpose was to explore the main AI techniques that could be used to construct a misuse or anomaly detection system. This review considered work both within and outside of the telecommunications context. The review was intentionally broad, and was developed prior to the receipt of extensive data samples and prior to extensive discussions on system requirements. As research progressed, and the requirements were formulated, more refined consideration of specific literature was possible, relating primarily to pre-processing of CDRs and development of neural techniques for anomaly detection.

b. *Evaluation of potential solutions.* An understanding of the project goals, gained through consortium discussions early in the project, and an appreciation of the nature and characteristics of CDRs, gained from the literature, allowed theoretical consideration of potential AI solutions to the FD/BP problem. From this, the need for a profiling approach emerged, and, based on the desire for generalisation capability, a neural solution was sought. Few options offer the ability to actually profile the user behaviour over an extended period, but through background research in different, but related, domains, a technique was identified, that could be adapted for this purpose. Given the lack of availability of usable data at the time, a generic solution was deemed appropriate. Various prototypes were then constructed (as described below), and evaluated on synthesised data sets. After initial evaluation of the feasibility, consideration of some preliminary CDR samples, and associated information, provided by Comarch, confirmed the suitability of the selected approach. Comarch played an important role in delineating the concerns of the FD/BP module within the overall MDS, and providing domain knowledge in the FD area, which in turn fed into the domain models.

c. *Development of domain models.* This was a key step in gaining understanding of the AI module context, and later clarifying the requirements, with the resulting models reported in the requirements document (D2.3.1), and the subsequent requirements update (D2.4.1). Following work on use cases, high level system decompositions, interaction models, *etc.*, two complementary modelling approaches were selected: behavioural process models (depicted as Role Activity Diagrams, RADs), showing the actions that are required from the domain from the major roles (either people or sub-systems), the interactions amongst those roles, and the dependencies and interdependencies of those actions; and a procedural model showing the system boundaries for FD/BP sub-systems, their input and output data, and those other major processes and systems with which they will exchange data. This model not only provides a context for the requirements, but also delineates the sphere of involvement for FD/BP, and shows where the data must cross sub-system boundaries. These models were again informed by discussions with Comarch, and by consideration of the high level business needs reported by Comarch and ERA in D2.1.1.

d. *Analysis of data.* Following intensive discussions concerning data formats, quantity and quality of data, and feasibility studies on the part of the operator (ERA), a sample set of CDRs for 500 subscribers over a one year period was provided. No accompanying information was provided by the operator, but early data mining and visualisation of various properties of the data revealed that some subscribers exhibited markedly different behaviour from perceived norms. Following discussions with the operator, it emerged that the data set did include examples of known fraudulent behaviour. This, then, allowed the investigation of various derived features, and the comparison between those features for sets of normal and abnormal data. Statistical models of the CDR data were derived, for gaining insight into the problem, as well as supporting future synthesis of test data. This ‘real’ data allowed the generic technique, previously developed based on synthesised data, to be verified as suitable for this specific domain.

e. *Development of prototypes.* Prototypes were developed initially for verification of the feasibility of the proposed approach, and later used also to support the analysis of the data samples. Various approaches were trialled, some based on data mining, others on anomaly and misuse detection. On receipt of the 500 subscriber data set from the operator, the prototypes were used to gain some indication as to the likely success for FD in particular. Results indicated by the AI approach as potentially anomalous were subsequently verified by the operator as true positive indications. Hence, the prototypes have been found effective for the identification of the fraud hidden in the data supplied by the operator. The remainder of the design process, and subsequent implementation, therefore focuses on refactoring the existing code base, development of the specified interfaces, and general software engineering concerns.

1.3 Artificial intelligence context

A plethora of artificial intelligence techniques have been applied to the problem of misuse detection (in the general sense), including *inter alia* expert systems, case based reasoning, artificial neural networks, evolutionary computation, swarm intelligence, artificial immune systems, and agent based systems, as well as a wide variety of statistical approaches to clustering, classification and regression. These have been reviewed in D2.2.1, which identified rule based systems as the dominant approach. This is also the approach currently used by the operator (PTC). Drawbacks with canonical rule based systems, of import in this domain, include:

- The need for a large knowledge base of rules for detecting known misuses, often requiring considerable investment in expertise and knowledge engineering.
- The lack of generalisation ability to unseen misuses, with such systems being built on the principle of misuse detection rather than anomaly detection.
- The lack of automated processes for adapting to new misuses and/or a changing environment.
- The interaction between rules in the knowledge base, which can make updating the system to cover a new misuse a non-trivial task.
- The need for house-keeping to purge the system of out-of-date or conflicting rules.

Contemporary AI techniques are a promising alternative, as they tend to offer, to varying degrees, the ability to learn, generalise and adapt, and, hence, address the following key issues:

- Automating the task of populating the 'knowledge containers' prior to system deployment.
- Enabling the generalisation of detection capability to unknown misuses.

It should be noted that these activities are not without user involvement. For example, the initial population of knowledge containers would be based on training data prepared by the user, and subsequent adaptation would require the system to be instructed when false positives or false negatives occur. The effectiveness of any AI based approach depends, to a considerable degree, on the nature of the data, not just in training but in use. AI based systems, explicitly or implicitly, essentially provide a mapping from some input space (e.g. features) to an output space (e.g. classes). When moving away from largely deterministic systems, it is worth bearing in mind the fundamental premises of the AI techniques involved; principally:

- That it must be possible to infer the output from the input, *i.e.* the data set being processed must have the appropriate information content to arrive consistently at the desired conclusions. This property is essential for learning. In classification terms, there is the concept of irreducible error, which is a measure of the extent to which the feature space does not permit the correct classification of samples (*e.g.* due to overlapping classes), regardless of the technique used.

- That similarity in the input space, however that might be measured, must map to similarity in the output space. Note that the converse is not necessary; different inputs could map to the same or similar outputs. This property is essential for generalisation, and the regularity of the input-output mapping will impact on the effectiveness of the system.

Whilst these may seem obvious preconditions, the nature of any data to be correlated / profiled for the detection of misuse needs to be examined to confirm that these are satisfied before AI techniques can be applied with any degree of confidence. In particular, the preparation of data samples, from which to derive the initial population of the knowledge containers, should consider the 'coverage' of misuse classes. For example, a misuse detection (as opposed to anomaly detection) mechanism is not likely to detect unknown misuses that are fundamentally different in nature from the known misuses on which it was trained. It should also be noted that the more 'flexible' architectures, such as neural and evolutionary systems, reduce the explanatory power of any detection system in which they are employed. A major advantage of canonical rule based systems is that they can present the user with the chain of reasoning by which they arrived at their conclusion. Other techniques, such as case based reasoning, can offer some explanation based on similarity measures, whilst others are often unable to present any indication of why they reached a particular conclusion. This is a natural dichotomy, rather than a weakness in the techniques - if one requires the flexibility to learn, generalise and adapt, then the determinism and explanatory ability of rule based systems must be sacrificed.

1.3.1 Learning paradigms

The objectives of FD and BP coincide, in that they both require a mechanism for profiling subscriber behaviour. For FD, the mechanism must be capable of raising an alarm when subscriber behaviour deviates significantly from learnt norms. For BP, the mechanism must allow the identification of similar behavioural patterns within a given data set. This mechanism is referred to, here, as a 'detector'. Hence, the objective of the learning strategy is to create one or more detectors capable of differentiation between normal and abnormal behaviour that may either be used for fraud detection or to classify subscriber behaviour. In a sense, once a detector is capable of fraud detection, it may also be used to support the business process aspects of the MDS, and, further, one may consider the potential of FD as the most significant of the two for revenue protection / optimisation, hence, the FD aspect forms the primary focus.

The success of a misuse detection system can be judged by two measures (although other forms are encountered within the literature):

- Detection rate
- False alarm rate

This may be decomposed into true positive (TP), false positive (FP), true negative (TN), and false negative (FN) indications, such that, if in a sample of size N there are X normal patterns and Y abnormal patterns, X + Y = N, then the detection rate is TP / (TP + FN), and the false alarm rate is FP / (FP + TN).

There are two primary learning strategies:

- *Learning from previous examples of misuse.* Somewhat confusingly this is often referred to as misuse detection, whereas, within the MDS documents, misuse detection is often considered in a more general sense. Adopting this strategy can lead to a system with poor generalisation ability (*e.g.* it cannot generalise to novel misuses that are 'significantly' different from the misuses on which the system has been trained. Thus, there is a tendency for the system to issue false negatives.
- *Learning from previous examples of normal behaviour.* This is often referred to as anomaly detection. The assumption is that patterns of behaviour that differ from those learnt from (assumed) normal behaviour constitute a misuse. This approach is particularly able to detect novel misuses, but, since it is difficult for a data set to capture the entirety of normal behaviour, this strategy tends to issue false positives.

Neither of the above approaches is entirely satisfactory. Hence, many hybrid systems employ a combination of techniques, some working on the misuse detection principle, others on the anomaly detection principle. It would not make sense for a contemporary correlation / profiling engine not to provide both functionalities, for deployment at the discretion of the end user. A key point is to manage the user's expectations regarding detection rates and false alarm rates. As with any AI system, the quantity and quality of the training data is paramount. This is equally true for both misuse and anomaly detection: in the former, for ensuring that the 'essence' of misuse is sufficiently captured to enable learning the true characteristics indicative of misuse and, hence, generalisable to new cases of misuse of the same class; in the latter, for ensuring that a sufficiently rich picture of normality is offered such that false positives are kept to manageable quantities. It is worth noting that training data is not necessarily static. That is, training should, arguably, be considered as an ongoing process, to reflect changes in environment, technology and subscriber behaviour.

In classification problems, particularly those characterised by small sample sizes but many features, the problem of 'spurious correlations' arises. There may be many ways of effecting the correct classification on the training set, without properly learning the underlying model. This is not a problem with the AI techniques; it may not be perhaps what would be in the mind of a human designer, but an AI based learning paradigm will infer what it will in order to try and solve the problem. It is possible to overcome this by incorporating additional domain knowledge, or by modifying the nature of the training data itself. Both require some *a priori* knowledge of the patterns that lead to a misuse, and, perhaps, properties of normal patterns. It is difficult to address if there is insufficient data, or domain knowledge concerning the data, such that an expert user could not identify, manually or automatically, when possibly inappropriate solutions have been learned.

In general, there may be many spurious correlations within a data set, which can only be detected and addressed by increasing the quantity and/or quality of training data, or through user involvement, where incorrect classifications on unseen data are subsequently used to adapt the training data or learning process. As the feature space increases, the probability of finding spurious correlations similarly increases. In this case, to allow the learning process to identify the true pattern of import, the quantity of training data must be increased until sufficient examples are provided such that it is not possible for the technique to correctly classify all examples without learning the true underlying relationship.

1.3.2 Training scenarios and domain knowledge

Whilst the quantity of data is important, and the requirements will vary depending on the nature of the data (such as the complexity of patterns to be detected), the degree of domain knowledge available for training will also impact on the nature and quality of the detectors created. Three broad scenarios are considered here, depending upon the level of domain knowledge available:

a. Labelled examples of normal and misuse behaviour.
b. Examples of normal behaviour only.
c. Examples of unknown behaviour.

Scenario (a) is ideal, and typical of supervised learning. The purpose of the learning process is to develop a detector that correctly classifies the training data as normal or abnormal (this could be divided into multiple classes, with each detector identifying one or more given misuse classes). Supervised learning is based on feedback on how well a given detector or set of detectors performs on the training data, which is often used to determine an error measure. Learning can be viewed as a search process, which seeks to find a combination of model variables that minimises this error. In the absence of domain knowledge, the learning process typically begins with one or more random detectors. It then uses the error measure to adapt these detectors with a view to minimising the error.

Scenario (b) is less ideal, but can be used to build an anomaly detection system, based on essentially unsupervised techniques. Such a system is trained to learn the essence of the input data such that deviations from normality can be detected by a difference in a response compared with that of normal data. The challenge, here, is to ensure that the training data offers sufficient coverage of normal behaviour, so that false alarms are minimised, and abnormalities are not present within the training data, so that a reasonable detection rate may be achieved.

If an anomaly is detected, it must be investigated by an expert to determine whether it is a misuse or simply innocent behaviour that was not represented in the training data. If confirmed, the misuse can be added to a database of known misuse scenarios (thereby building a database that could later be used for supervised training). If unconfirmed, the detector may be discarded and, possibly, the training data updated to prevent the subsequent creation of further incorrect detectors. A strategy for maintaining the training set(s) is required to prevent endless expansion.

Unlabelled data, scenario (c), requires an unsupervised learning strategy; there is no information that could be used to guide a conventional learning approach. Techniques such as clustering, dimensional scaling, feature maps, and regression, are used to explore patterns within the data. This is generally referred to as data mining. The underlying assumption is that the patterns observed by such techniques (if, indeed, patterns are identified) are somehow related to normal and anomalous behaviour. That is, anomalous behaviour is statistically different from normal behaviour. Where this does not hold, the data mining will not be able to identify patterns of use in this domain. Further, patterns may be identified based on other characteristics that are not of interest to the problem of misuse. Hence, the success of such a process is highly unpredictable. Patterns identified within the data have to be examined by domain experts to ascertain what, if anything, they may mean in the context of the domain. This is, therefore, a purely offline process. On the other hand, where there is insufficient knowledge of the domain, data mining and subsequent expert analysis might lead to identified misuses that can then be incorporated into a training set for supervised learning to producing detectors for online monitoring of activity.

In all scenarios, it must be possible to distinguish between normal and abnormal behaviour based on the features available. If a clean separation is not possible, misclassification will occur. Further, when considering profiling of behaviours over an extended period, it is possible that a small number of unusual calls within the period may be outweighed by the remaining normal behaviour and not show up as an anomaly.

1.3.3 Generalisation

The benefit from an AI approach comes not only from automating the learning (knowledge discovery) process, but also from generalisation. In scenario (a), one would hope that a trained detector would detect not only the specific misuse examples included within the training data, but also misuses that are ‘similar’ in nature. In scenario (b), one would hope that a trained detector will have learned the essence of normality such that any deviations from learned norms are detected. Not all techniques within the broad field of AI offer generalisation capabilities. Neural networks are the obvious example of a paradigm that does, and it is advantageous, for the maximisation of detection rates, that generalisation is possible. Even with such a technique, training is a difficult process to get right, and the notion of over-fitting, where the detector effectively develops a look up table for the specific training data and shows little ability to respond correctly to unseen data, becomes a relevant concern.

An automated learning system potentially offers other, less obvious, benefits. For example, it may avoid the implicit bias that expert analysts may bring when developing, say, a rule based system, and thus form a completely different (possibly more effective) view of the problem. Further, whilst experts are limited, in general, to capturing the essence of relatively simple relationships amongst a small set of variables, an automated learning system may effectively process a larger feature space, and extract more complex relationships (derived knowledge).

1.3.4 Adaptation and User Intervention

AI engines are often employed within dynamic environments. That is, environments in which the patterns of interest may vary with time. It is also likely that the learned patterns are not completely reliable, and give rise to either/both false positive or false negative indications. Hence, it is necessary to consider mechanisms for user intervention and adaptation. In cases where there is only partial knowledge of misuses, user intervention is essential.

On identification of a misuse, the relevant portion of the source data must be processed by an expert analyst to verify that a misuse has occurred. If it is found to be a false positive, the case should be added to the training data and the detectors retrained to prevent further occurrence of the false alarm. Note that a false alarm could be raised through either poor training/generalisation or a change in the environment such that previous misuse patterns no longer accurately indicate misuse. Similarly, misuses may be identified by an expert analyst that the AI engine overlooks. The anomaly detection approach might be useful in dynamic environments to pick out particular patterns that the system sees as differing from normal behaviour. Whether this results in an updated view of normality or a new misuse detector, user intervention is required. Data mining may be employed in an offline process to identify patterns of interest in the data. These are then explored by an expert analyst to determine whether any such patterns are indicative of misuse. Where this leads to improved understanding of normality and/or abnormality, the user must be able to update training sets and retrain existing detectors or, where anomaly detection techniques are used, the database of normal behaviour needs to be updated to prevent incorrect novelty detectors from being produced.

1.3.5 Summary and recommendations

The available data will be the single most significant determinant of AI technique selection. However, regardless of the initial training scenario, over time, knowledge will be extracted that enables one to move from, say, data mining to anomaly detection, or from anomaly detection to misuse detection. Thus, an effective solution to this problem requires a hybrid approach that supports this process, and can be refined and improved in deployment.

From the preceding discussion, the following expectations emerge:

- Misuse detection, given examples of known misuse behaviour.

- Anomaly detection for unknown misuses based on examples of normal behaviour.
- Offline data mining process for identifying candidate behaviours for expert analysis.
- Selection of techniques with the ability to generalise from learned behaviours to unseen behaviours.
- Support for user intervention, for highlighting incorrect indications, adjusting training data and retraining detectors.

The latter concerns management of the 'knowledge repository' and is covered by the Decision Support Module (DSm). All other concerns are catered for by the FD/BP approach.

1.4 Overview of the approach

The preceding section has outlined both the expectations for an AI approach to misuse detection (in the general sense), as well as constraints / issues related to the domain knowledge available, especially where it affects the quality and quantity of training data. In this section, the means of addressing these expectations is outlined.

1.4.1 Building domain knowledge

The primary concern in designing the FD/BP module was a lack of domain knowledge. Following intensive discussions with the operator, and their subsequent feasibility studies, it was found that sufficient (*e.g.* in terms of quality, quantity, or domain knowledge) training data could not be provided to support a misuse detection approach, since only one misuse class was known. For anomaly detection, there is sufficient training data, if certain assumptions are made concerning normality. Data mining is possible, but verification of found patterns requires expert involvement and is a time consuming process. Hence, the challenge can be summarised as being one of moving from a situation of minimal domain knowledge, to building a repository of derived knowledge. This process may take considerable time, but should be supported by the approach. That is, first one may consider the identification of patterns using data mining techniques. These could help in isolating seemingly normal behaviour under certain assumptions (*e.g.* that normal behaviour is more prevalent than abnormal behaviour). Since fraud is estimated to run at somewhere in the region of 3% of turnover within the mobile telecommunications market, this is considered a safe assumption given a reasonably large random sample of data. Gaining confidence in the normality of a given set of data enables an anomaly detection approach to be deployed. This involves the training of one or more detectors on the presumed normal data, and then using these to identify when current behaviours deviate markedly from learnt norms. Such deviations are then subjected to expert investigation, to ascertain whether they represent:

a. Fraud or other misuse cases.

b. Non-fraudulent behaviours of interest from the perspective of the operator's business processes.

c. Non-fraudulent behaviours of no interest to the operator.

In case (a), in addition to responding to the fraud or other misuse problem directly, one can start to build a set of known cases (*i.e.* labelled data) to support the training of future detectors following a supervised (misuse detection) learning paradigm. This may generally be considered to offer the maximum potential for the detection of future misuses of the same or similar nature. Similarly for case (b), although it may be beneficial to separate such concerns from misuse detection. Case (c) represents a false positive indication, for which some adaptation of the anomaly detector may be considered desirable, particularly if many false alarms are being generated. Such adaptation may be accomplished by modification of the training data and retraining of the detector. Hence, there is a need to retain training data for subsequent adaptation. This is within the scope of the DSm.

The mechanisms by which domain knowledge may be gradually increased, with the support of domain experts, is depicted in Figure 1. Expert involvement is denoted by the dashed lines, whilst processes relating to the provisions of the MDS are denoted by solid lines. As stated in the requirements document (D2.4.1), the data mining will be supported by existing open source data mining tools (such as *Weka*), whilst anomaly detection and misuse detection will be implemented within the AI module itself.

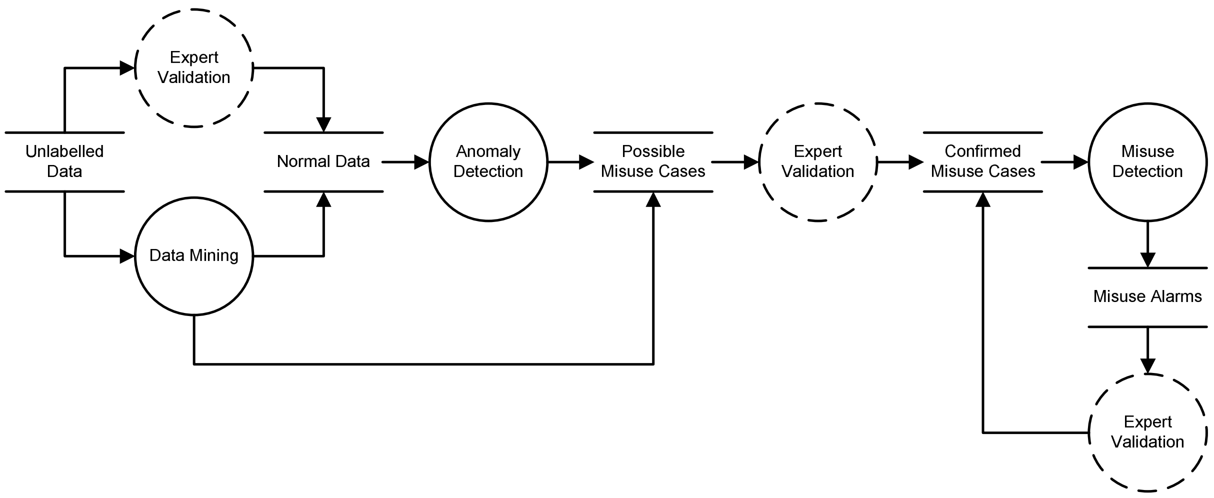


Figure 1. Process of building domain knowledge from initially unlabelled data.

1.4.2 Artificial intelligence based solution

The solution to be implemented within the MDS is depicted, at a high level, in Figure 2. Sets of Call Detail Records (CDRs) are processed to extract features that enable subscriber behaviours to be profiled. In training, an unsupervised neural network is developed by an evolutionary learning algorithm (a hybrid of genetic algorithms and particle swarm optimisation). The training process results in a network (or detector) that, for a given set of input CDRs, indicates the extent of deviation from the behaviours present within the training data. This deviation is a continuous (dimensionless) output that may then be thresholded to generate alarms, or used to prioritise investigations. Two subsystems are therefore required: training, based on historic data, and detection, typically based on online data.

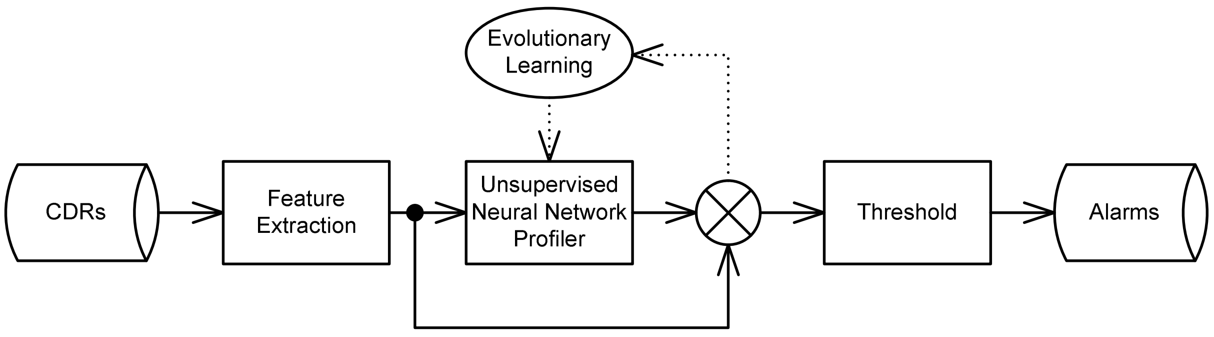


Figure 2. Artificial intelligence based solution to meet the FD/BP requirements of the MDS.

The expectations, outlined in Section 1.3.5, are addressed by this solution, as follows:

- *Misuse detection.* By training the profiler on data containing instances of known misuses of a given class, deviation in the output of the profiler is indicative of normality. This is a novel approach, insofar as one would normally attempt to train an AI engine for misuse detection on both positive and negative examples. However, this approach as two advantages: first, it allows the detection of specific misuses by specialised detectors, without the need for different techniques to be implemented; second, it avoids the need for extensive examples of normal behaviour to be captured, since only the misuse cases will be used for training.
- *Anomaly detection.* As above, but trained in the conventional manner, on examples believed to represent normal behaviour.
- *Data mining.* In addition to conventional data mining techniques, an approach has been developed that clusters the output of one or more detectors in order to identify groups of subscribers exhibiting similar behaviours. This is described in more detail in Section 1.6.4.
- *Generalisation.* The use of neural networks as the underlying profiling technique allows the generalisation from learnt behaviours to unseen behaviours.

General features of the approach, include:

- No specific domain knowledge is assumed, nor are any particular characteristics of the data (e.g. no assumptions concerning distributions).

- Neural network profiler extracts the 'essence' of the calling behaviours from the descriptive features, which in turn are derived from the CDRs.
- Detects deviations from learned behaviours – the learned behaviours may represent either normal or abnormal usage, *i.e.* anomaly detection or misuse detection.
- Produces a continuous output that can be thresholded to generate alarms of varying severity, or used to prioritise investigations.
- Output represents changes in subscriber behaviour that may be indicative of fraudulent usage or changes in normal calling patterns.
- Potential to detect both known and unknown frauds.

1.4.3 Feasibility study

As highlighted, above, work within the MDS domain suffers from a lack of well understood and pre-classified data samples. Further, due to the timescales involved in the operator providing the 500 subscriber set used for verification of the approach, a generic technique was needed, which had to be developed without knowledge of the specific properties of the domain. Fortunately, the problem of anomaly detection is fairly generic in nature: given a set of normal feature vectors, is it possible to extract an underlying model that sufficiently reflects the essence of those features, such that abnormal feature vectors may be detected (where abnormal is defined as being something markedly different from the normal sample, rather than specifically fraud).

A wide variety of test data can be found to support such work, either based on corpora from other domains, or data which has been synthesised by random sampling from various probability distributions. Here, the functionality of the AI approach is illustrated using random profiles generated from normal probability distributions[1], according to the specifications in Table 1 and Table 2. Three data sets are defined:

a. Training set. The only data used within the evolutionary learning process.
b. Control set. Data of similar nature to training set, used to verify that the detector has been properly trained.
c. Test set. Representing data with different properties that the training and control sets.

[1] It is not suggested that normal probability distributions do, in fact, reflect the real distributions of telecommunications data. In reality, one would expect a Poisson process to be a reasonable model of individual events, and, thus, an Erlang distribution (equivalent to a Gamma distribution with integer values of the shape parameter) to be a reasonable model of the distribution of call frequency and call duration. However, since this section is not concerned with the nature of the distributions within the application domain, but in the principle of anomaly detection using evolutionary-neural computing techniques, this is considered a reasonable approach for an initial feasibility study.

The data sets each comprise six attributes, similar in nature to some of the features expected to be extracted from CDR logs, and are considered reasonable for profiling different subscriber behaviours (as described in the literature). As can be seen from the tables, the properties of the test set differ markedly from that of the training and control sets. If the technique is functioning as an effective anomaly detector, one would expect the following behaviour:

a. Low values of deviation on the training set. Such deviations reflect the variation present within the training data and the extent to which the underlying model can accurately predict the expected output.

b. Similarly low values of deviation on the control set. In this case, identical distributions are used, so the similarity should be clear. In practice, there may be more variability in the input data, so further deviation may be expected.

c. Markedly higher values of deviation on the test set. This is generated from distributions that are clearly different from the training set, and, hence, should appear to be distinguishable as anomalous.

This is simple test of the approach, and one might question why it is important. The point is that only the assumed normal data from the training set is used during the learning process. For the AI engine to work as an effective anomaly detector, it needs to be capable of extracting the essence of this normal data, and, thus, be able to recognise data drawn from different distributions as deviations from normal. The output of the AI engine is illustrated in Figure 3. As can be seen, the output on the training set, although not showing zero deviation (reflecting the natural variation within the data), is low, and the control set shows the required similar behaviour. The test set is clearly showing up as markedly different from the learned norms. A simple threshold may be set to differentiate between normal and abnormal data. The threshold may be determined by considering the peak deviation on the training data and allowing some additional margin.

Attribute	**Type**	**Mean**	**Std. Dev.**
# calls	Integer	40	10
Average call duration	Real	10	3
Ratio of weekday calls to total calls	Real	0.75	0.1
Ratio of calls during working hours to total calls	Real	0.9	0.05
# destinations	Integer	25	10
# base stations	Integer	6	2

Table 1. Attributes and distribution parameters for training and control data sets.

Attribute	Type	Mean	Std. Dev.
# calls	Integer	100	20
Average call duration	Real	30	10
Ratio of weekday calls to total calls	Real	0.5	0.2
Ratio of calls during working hours to total calls	Real	0.5	0.2
# destinations	Integer	50	10
# base stations	Integer	20	5

Table 2. Attributes and distribution parameters for test data set.

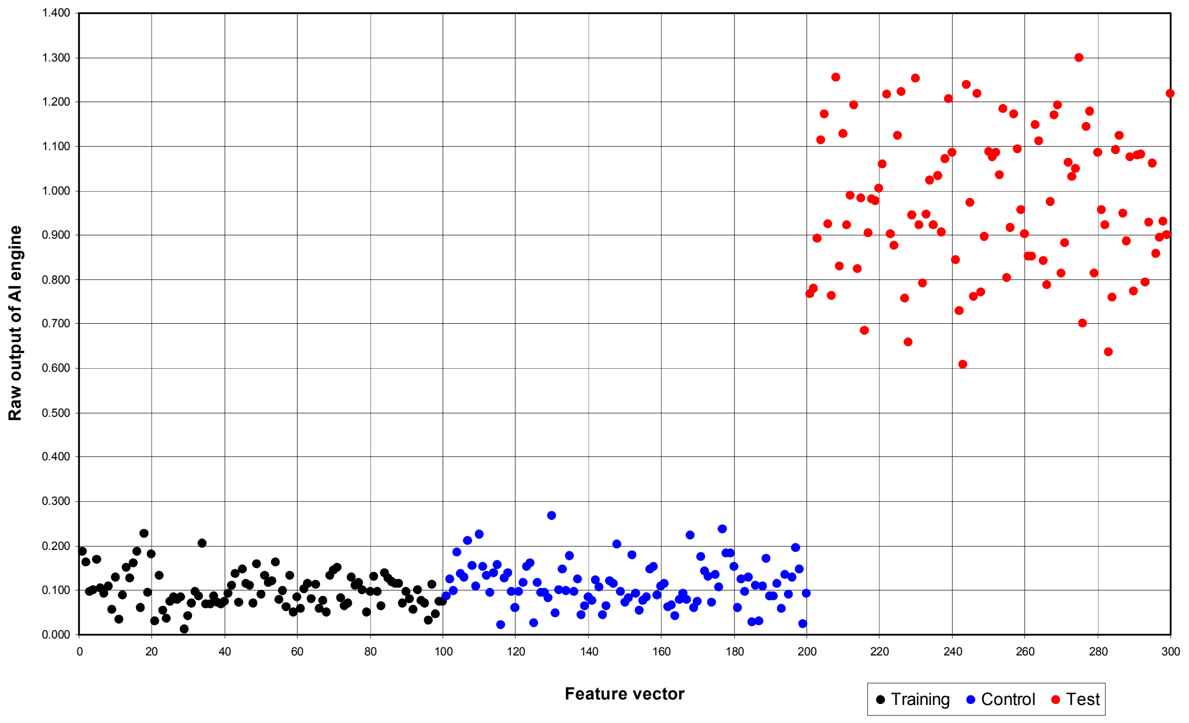


Figure 3. Output of the AI engine for the three synthesised data sets.

1.5 CDR data samples

Call Detail Records (CDRs) may contain a variety of information. Those provided by the operator include the following fields:

- Phone number
- Other party phone number
- Direction (to network, from network, forwarded)
- Call start date
- Completion status (completed, dropped)
- IPS user label (base station)
- Third party phone number
- Call duration

CDRs for 500 subscribers, collected over a one year period, were provided by the operator for the analysis and prototyping of the FD/BP module. Since a single call does not carry much information about overall subscriber behaviour, it is difficult to assess whether a given call represents a misuse or other behaviour of interest. Hence, there is a need to establish 'profiles' of subscriber behaviour. Thus, features are derived from a set of CDRs, which are more appropriate for representing calling behaviour over a period of time. In the prototype system, this period has been set to one week. It was confirmed by the operator that weekly responses would be adequate. A weekly period has the benefit of eliminating challenges due to variations caused by, for example, markedly different calling behaviours on weekdays and at the weekend. However, the assumption of weekly periods has not been built into the AI engine, and shorter periods may be adopted if needed (although, it might then be better to treat weekend and weekday profiles separately). A further advantage of considering whole weeks is that there is generally sufficient usage to enable a profile to be built. Typical usage is approximately 33 calls per week, whereas this would amount to a small number of calls per day, and a high degree of variability (noise) in, for example, daily profiles.

1.5.1 Features

From each (nominally weekly) set of CDRs, a set of derived features, which are potentially useful for building a profile of subscriber behaviour are produced. Four general classes of feature are defined:

a. *Volume*. Information related to the number and duration of calls.
b. *Temporal*. Information related to times of the day and days of the week when calls were made or received.
c. *Destination*. Statistics about the destinations accessed.
d. *Location*. Statistics about the base stations used.

The feature set comprises:

- Call duration statistics
  - Total number, total duration, mean, median, standard deviation, inter-quartile range
  - Duration histogram with bins:
    - < 5S
    - 5-10S
    - 10-60S
    - 1-5M
    - 5-10M
    - 10-30M
    - 30-60M
    - >60M

- Calls completed and calls dropped
- Daily counts
    - Calls in each of 24 bins for each hour of each day
    - Calls during daytime (8:00-17:00), evening (17:00-0:00), and night (0:00-8:00)
- Weekly call counts, as per daily call counts, aggregated per week
- Weekend counts and weekday counts
- Base stations
    - Total used during week
    - Histogram of 10 most frequently used
- Destinations
    - Total number of destinations
    - Histogram of 10 most frequently used

Note that all features are replicated for all calls, incoming calls, outgoing calls and forwarded calls. By capturing information pertaining to call volumes, destinations, mobility, *etc.*, these features allow discrimination of various subscriber behavioural patterns. The selected AI approach is flexible and would allow the addition of further features derived from the CDRs.

1.5.2 Properties

Initial investigation of the data set focussed on various visualisations of properties such as call volume, destination distribution and base station distribution. Through this process, it was discovered that, from the total 500 subscribers in the data set provided by the operator, the last (in terms of the order in which the subscriber first appears in the source files) 200 or so, appear to contain extreme behaviours, buried within other seemingly normal behaviour. The operator verified that these were indicative of a particular class of fraud (FCT). This is illustrated clearly in the bubble plot of Figure 4. This particular plot illustrates call frequency, but call duration also shows similar anomalies. Taking this information, it was possible to explore which features reflected the fraudulent usage. Many plots were generated between pairs of features, for various parts of the data set. This is illustrated in Figure 5, where the supposed normal portion of the data set (the first 300 subscribers) and the supposed abnormal portion of the data set (the last 200 subscribers) are distinguished by colour: blue for normal, red for abnormal. This figure examines call durations, call counts, destination numbers, weekday and weekend calls, and base stations used. The interesting points to note are:

a. That the anomalous behaviour is indicated across numerous features, and therefore detection involving a range of features may be more reliable in terms of enhancing class separation. The current rule based approach for detecting FCT fraud considers call volume measures only. Here it is seen that the distribution of destination numbers and the mobility are also strong indicators.

b. There is an overlap between the normal and abnormal data, since (as seen in the bubble plot) the 200 subscribers within which the fraudulent behaviour is buried, contain many weeks of otherwise normal looking behaviour.

c. In general, the FCT fraud is characterised by a combination of higher call counts, longer calls, more even distribution of calls throughout the week, and less mobility.

d. Such analysis may be useful, not just for developing AI solutions, but for improving rule based approaches, by highlighting the potential for combining (probably fuzzy) rules related to a number of attributes, rather than just call volume.

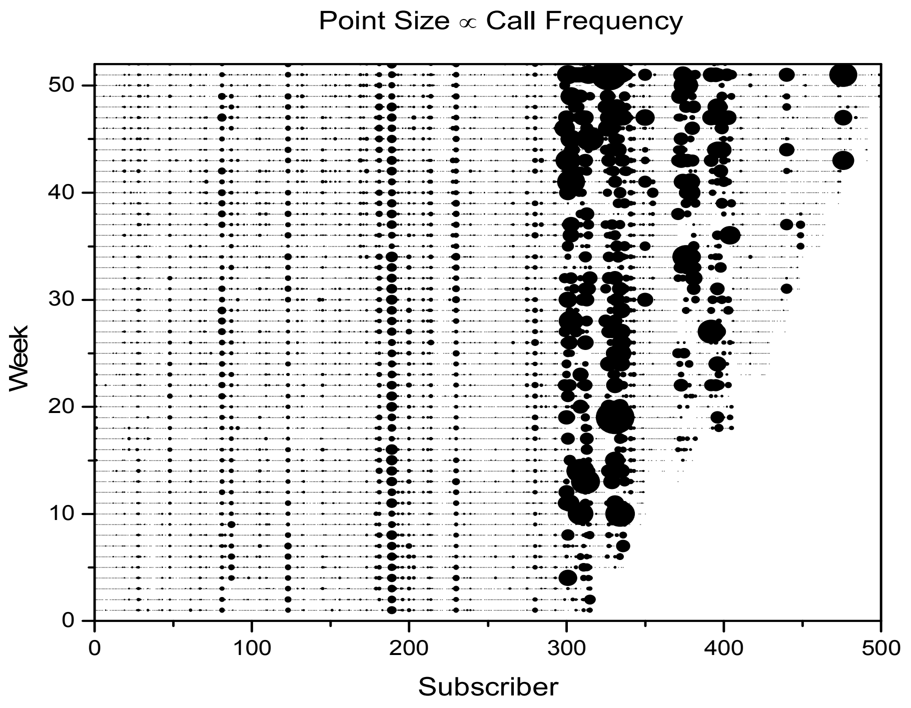
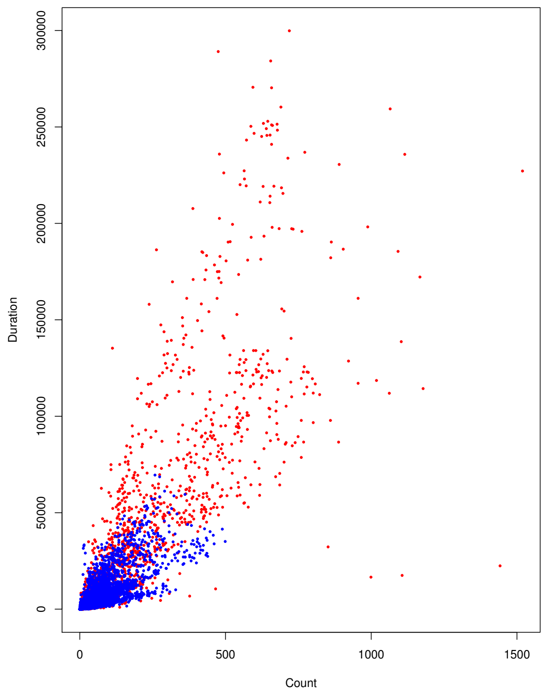
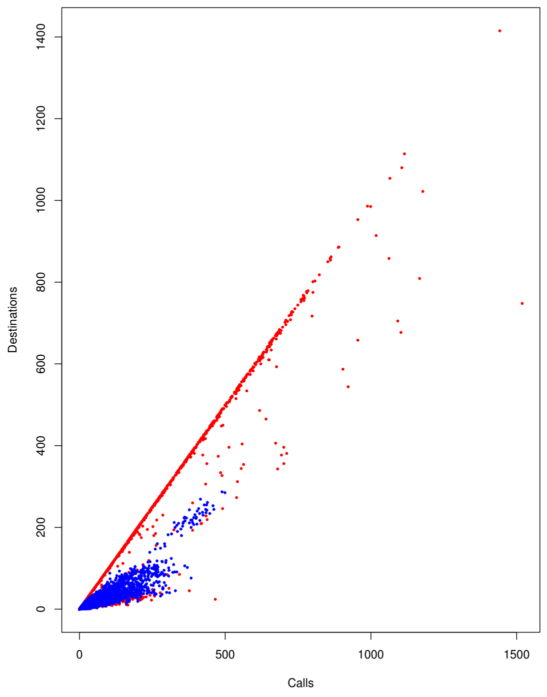


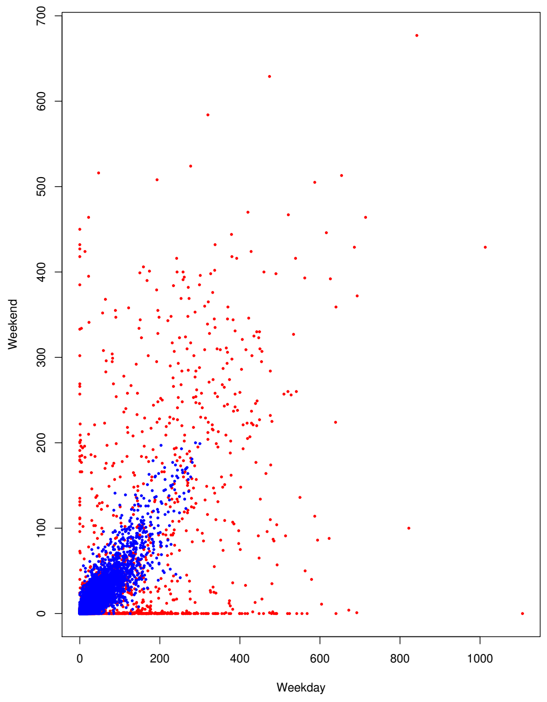
Figure 4. Bubble plot showing call frequency per subscriber per week. Note the many blank weeks, particularly in the latter half.



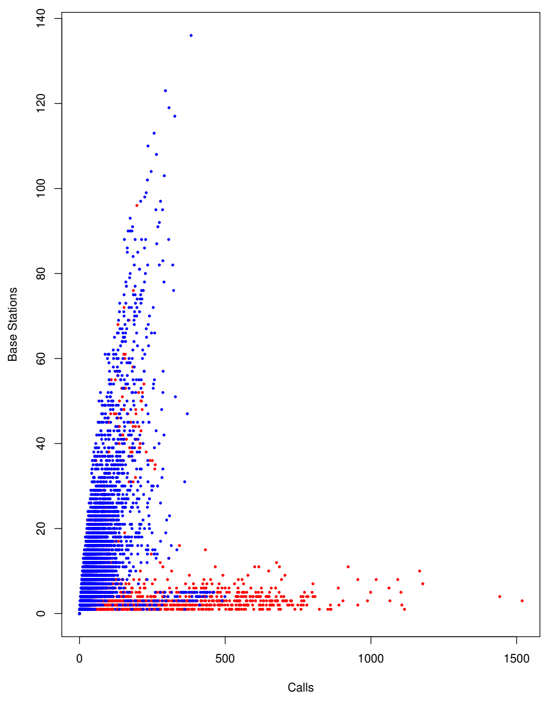
(a) Call duration versus call count



(b) Destination number versus call count



(c) Weekday versus weekend call count



(d) Base stations versus call count

Figure 5. Scatter plots of various feature pairs. Blue points refer to the supposed normal data. Red points refer to data with hidden fraud cases.

1.5.3 Data analysis

In this section, a summary of some of the statistical properties of the CDR data is provided. Figure 4 illustrates the call frequency within the 500 subscriber data set provided by the operator. The clear extreme behaviour corresponds to the 'hidden' cases of FCT fraud. The statistical properties reported here are taken from the first 250 subscribers, and are therefore considered a reasonable representation of normal behaviour. The records of the 250 subscribers, over a 52 week period, comprised 428876 individual CDRs, and, hence, an average of approximately 33 calls per subscriber per week. An understanding of the underlying properties of the data is useful, particularly in terms of synthesising test data. Hence, this analysis is presented largely from the perspective of characterising assumed normal behaviour, to support such synthesis, where it might be required during the testing and evaluation phase of the project.

When statistics are collected for CDRs in timescales of less than one week, it is not clear whether the activity is consistent with a more long term customer profile, *i.e.* the statistics may not follow the global distribution. In particular, there is some difficulty in resolving the potential differences in calling behaviours on weekdays versus weekends, and this may well require two sets of statistics, or different statistics for different groups of subscribers. Hence, to avoid such concerns, features are considered over one week periods. From the raw data, an aggregated set of records was produced that represented weekly call activity for each customer. A total of seven weeks that had zero calling activity were removed, affecting five subscribers.

1.5.3.1 Volume

This section considers features pertaining to call volume. The first issue to be considered is that of modelling the total call counts, and their division into sent and received calls. Figure 6 shows a scatter plot of sent versus received call counts.

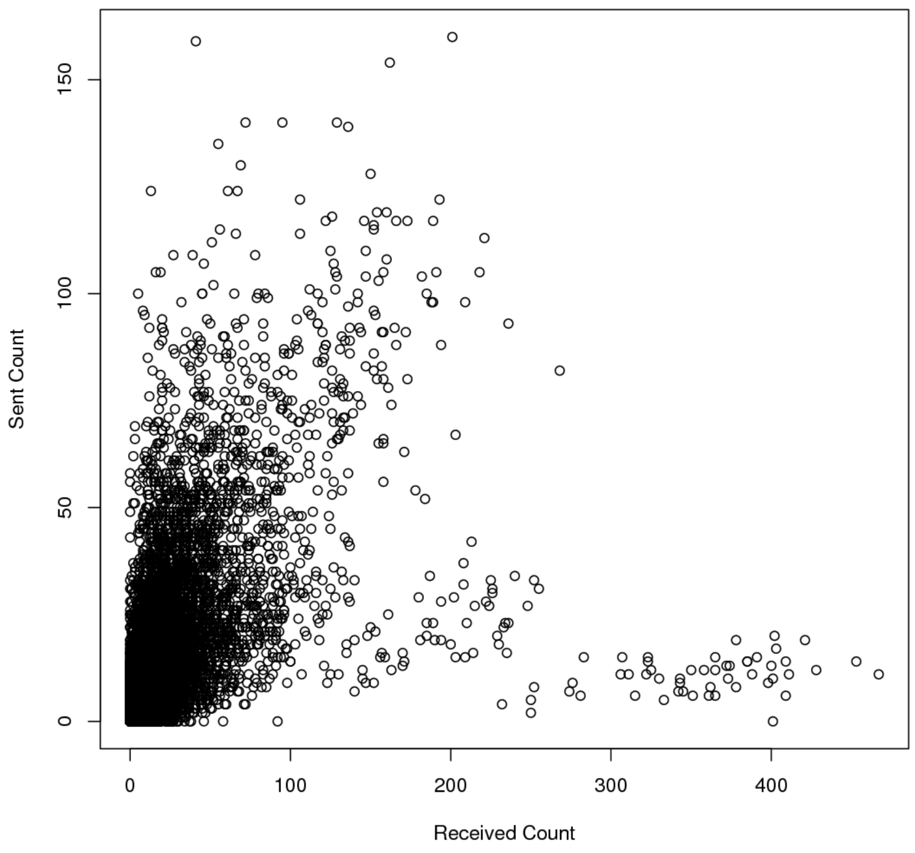


Figure 6. Scatter plot of counts of sent calls versus received calls.

Visual inspection does not suggest an obvious correlation between these attributes, which have a correlation coefficient of 0.4663. There do appear to be some distinct patterns of behaviour within the data set, with, for example, a subset of records that show a large number of received calls but relatively few sent calls. There is not, however, any reasonable basis for the removal of these points from the data set. It is assumed, therefore, that no direct relationship exists between the number of calls sent and those received.

Thus, for synthesis, a simplified approach is possible, which aims to maintain an approximately constant ratio of sent and received calls. This ratio is obtained from the raw data and is represented as the ratio of the means. The mean number of calls sent is 14.27 and the mean number of calls received is 19.68, that is, 57% and 43% respectively. The actual distribution of sent and received call counts is illustrated in Figure 7 and Figure 8, for sent and received calls respectively.

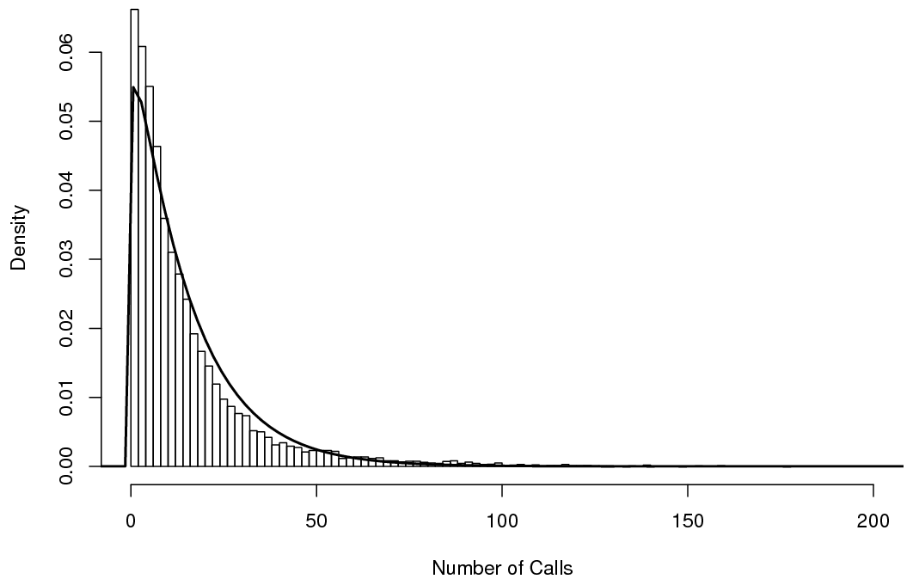


Figure 7. Distribution of calls sent.

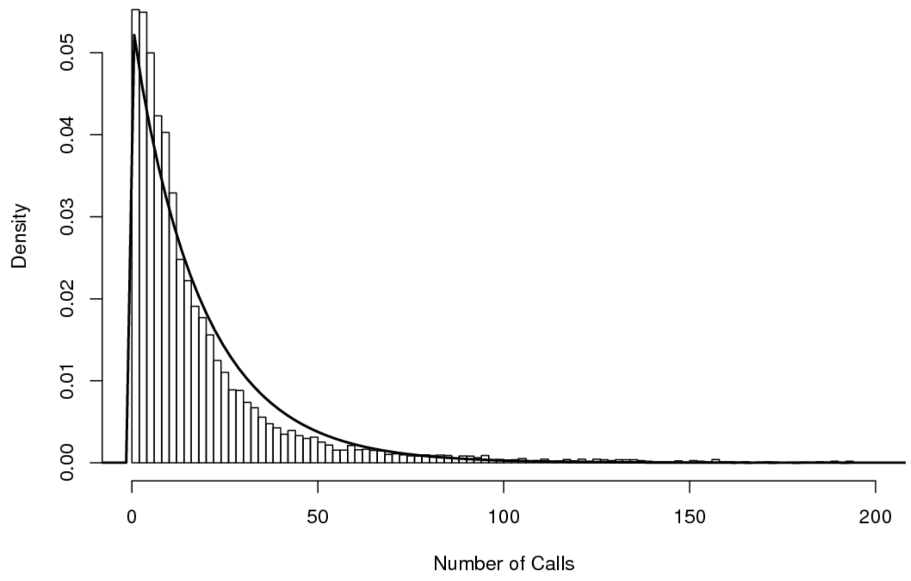


Figure 8. Distribution of calls received.

The histograms of Figure 7 and Figure 8 are modelled here using a Gamma distribution. Not only is this a reasonable fit to the data, but, typically, events in communications systems are modelled by a Poisson process, and hence the distributions are appropriately modelled by a Gamma distribution (or, alternatively by an Erlang distribution, which has been used to model traffic load in telecommunications systems, and is a special case of the Gamma distribution, with integer values of the shape parameter, α). The Gamma distribution can model highly kurtotic distributions and follows exponential decays either side of a maximum. A random variable has a Gamma distribution if the probability density function is given by:

$$f(x|\alpha,\beta)=\begin{cases}\frac{\beta^{\alpha}}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x} & \text{for } x>0\\ 0 & \text{for } x\le 0\end{cases}$$

A random variable following the Gamma distribution has the following properties:

- mean $\frac{\alpha}{\beta}$
- mode $\frac{(\alpha-1)}{\beta}$ for $k \ge 1$
- variance $\frac{\alpha}{\beta^2}$

Thus, certain calling behaviours may be characterised by the two parameters α and β. The gamma distributions are overlaid on the histograms. For sent calls, the parameters are α = 0.0697 and β = 1.0750, and for received calls α = 0.0523 and β = 0.9890. Having modelled the counts, the duration must then be considered, with again attention to the possibility of correlations between attributes. Figure 9 illustrates the duration of calls sent versus calls received.

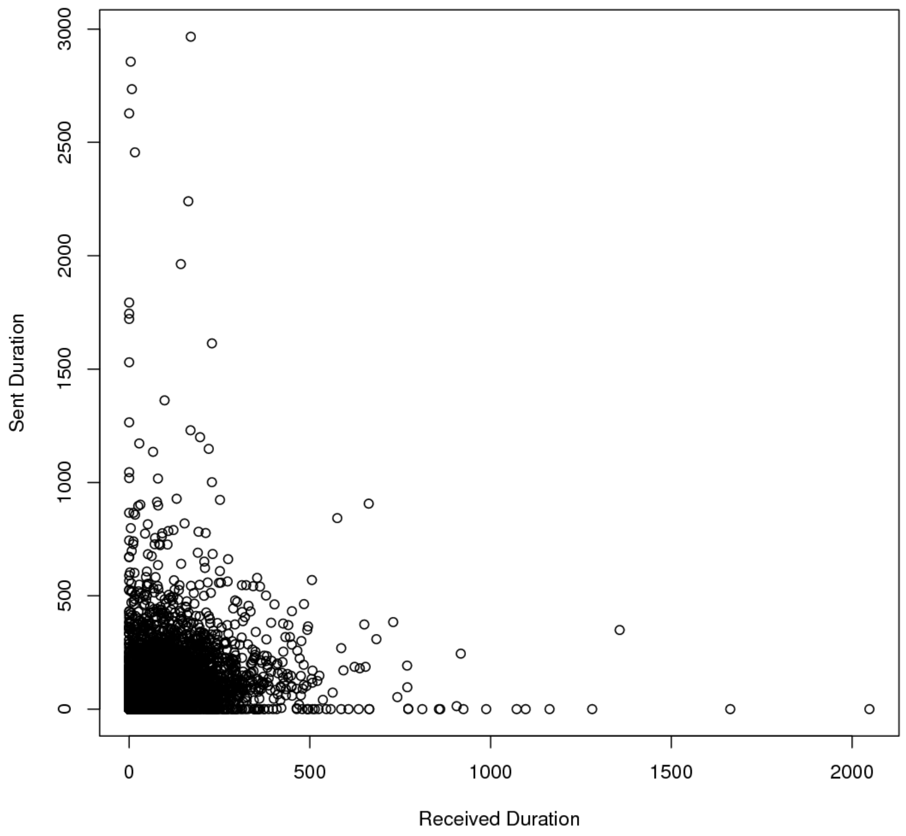


Figure 9. Sent and received call duration.

Visual inspection does not suggest any correlation between sent and received call durations, and this is supported by a correlation coefficient of 0.1748, and they can, therefore, be considered independently. The histograms for sent and received call durations are provided in Figure 10 and Figure 11, respectively.

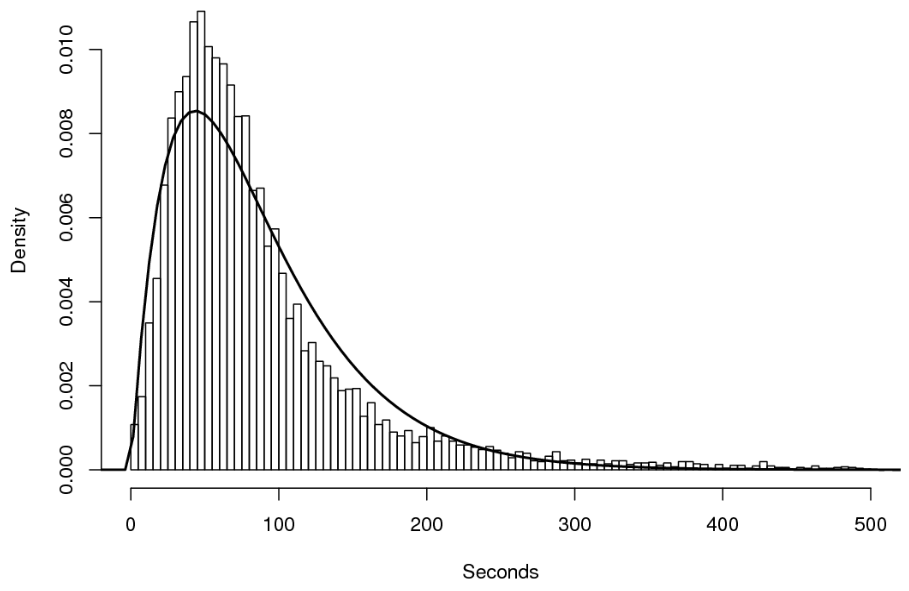


Figure 10. Distribution of call duration for sent calls.

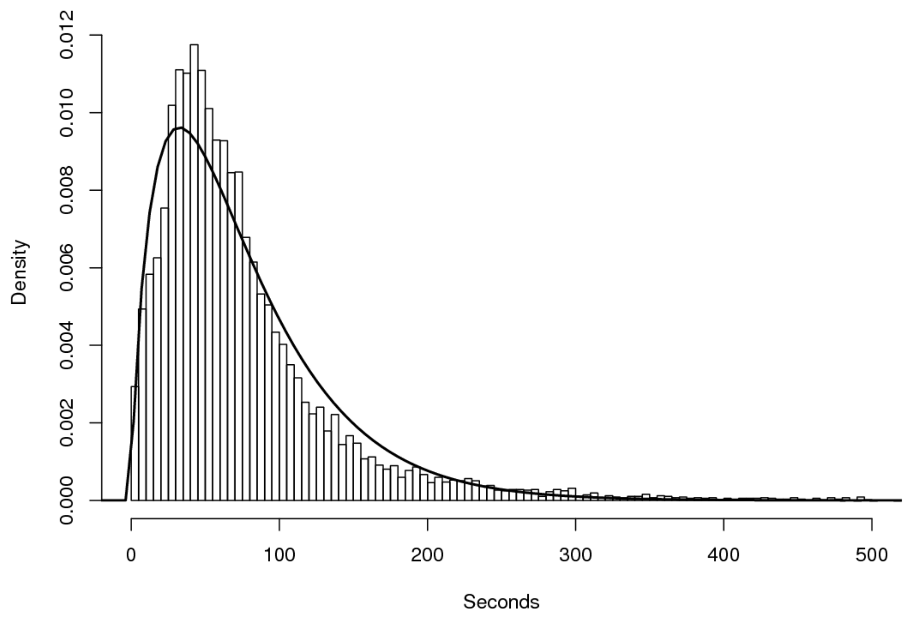


Figure 11. Distribution of call duration for received calls.

For sent calls, the parameters are α = 0.0234 and β = 2.0180, and for received calls α = 0.0233 and β = 1.7558.

1.5.3.2 Temporal

With the total call count, and sent and received call distributions, established, the next concern in terms of the synthesis of representative data is to establish an appropriate distribution of these calls over the time of the day, and the split between weekend and weekday. This is achieved by establishing appropriate ratios into which the calls may be divided. Figure 12 shows the distribution of sent and received calls by time of day, with hourly bins.

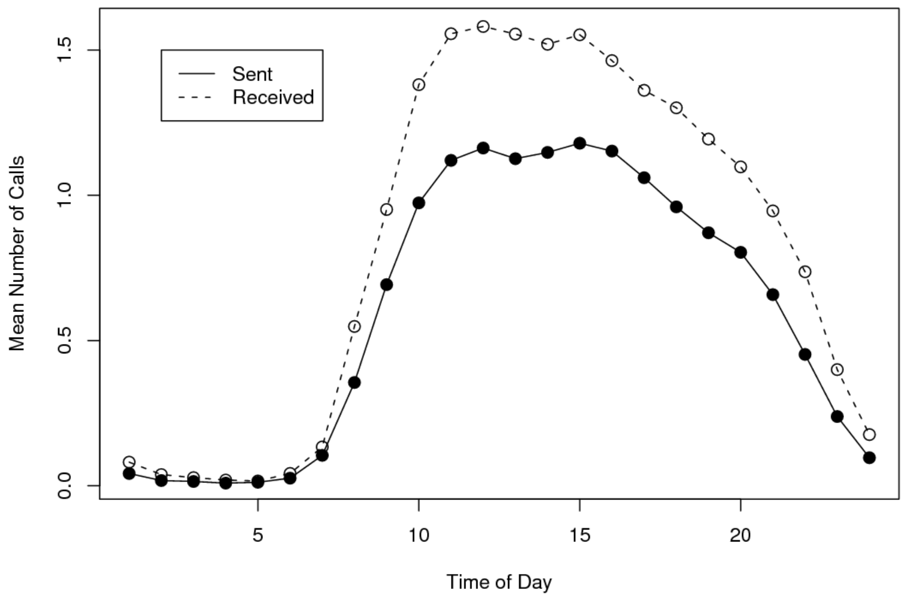


Figure 12. Distribution of calls by time of day.

Rather than model this somewhat complex distribution, a simplified approach, and one that is in line with existing research, is to divide the day into three periods and count the calls in each (daytime, evening and night), and to consider the split between weekend and weekday calls. The average call counts and the proportion of calls within each daily period are given in Table 3, and for weekend and weekdays in Table 4.

	night (0:00 – 8:00)	**daytime (8:00 – 17:00)**	**evening (17:00 – 0:00)**
Average number sent	1.274	8.922	4.079
Average number received	1.859	11.970	5.851
Proportion sent	0.09	0.63	0.28
Proportion received	0.09	0.61	0.30

Table 3. Call counts and proportions within each daily period.

	weekday	**weekend**
Average number sent	9.75	4.524
Average number received	13.437	6.243
Proportion sent	0.68	0.32
Proportion received	0.68	0.32

Table 4. Call counts and proportions between weekdays and weekend.

1.5.3.3 Destinations

Figure 13 and Figure 14 show the number of different destinations against the number of calls, for sent and received calls, respectively.

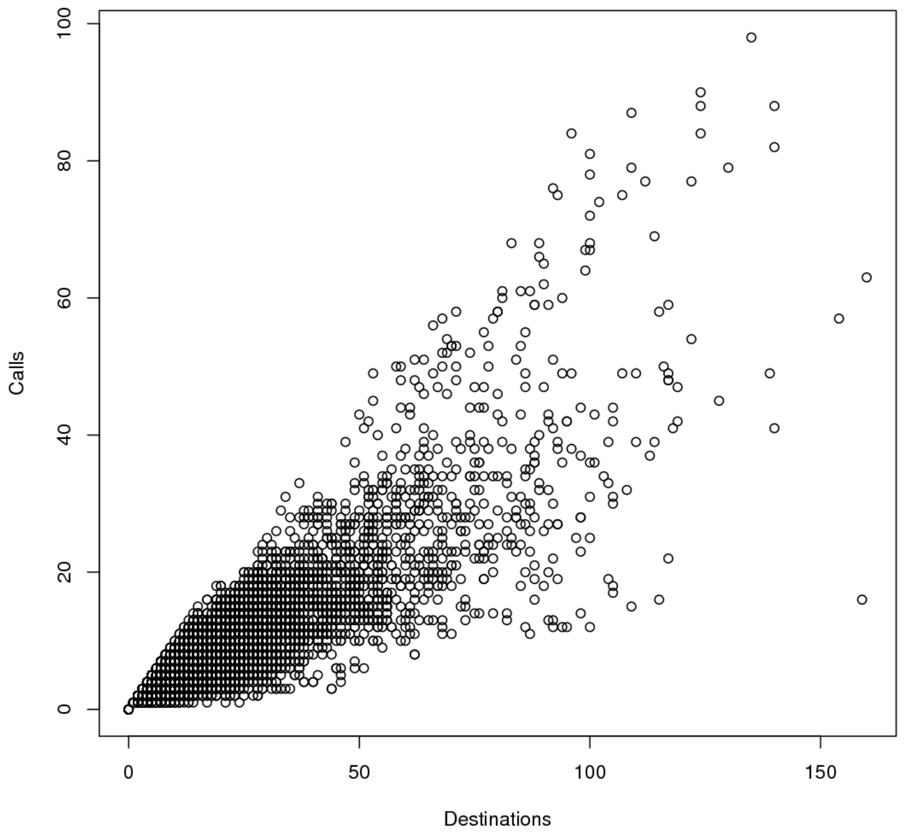


Figure 13. Distribution of destinations for sent calls.

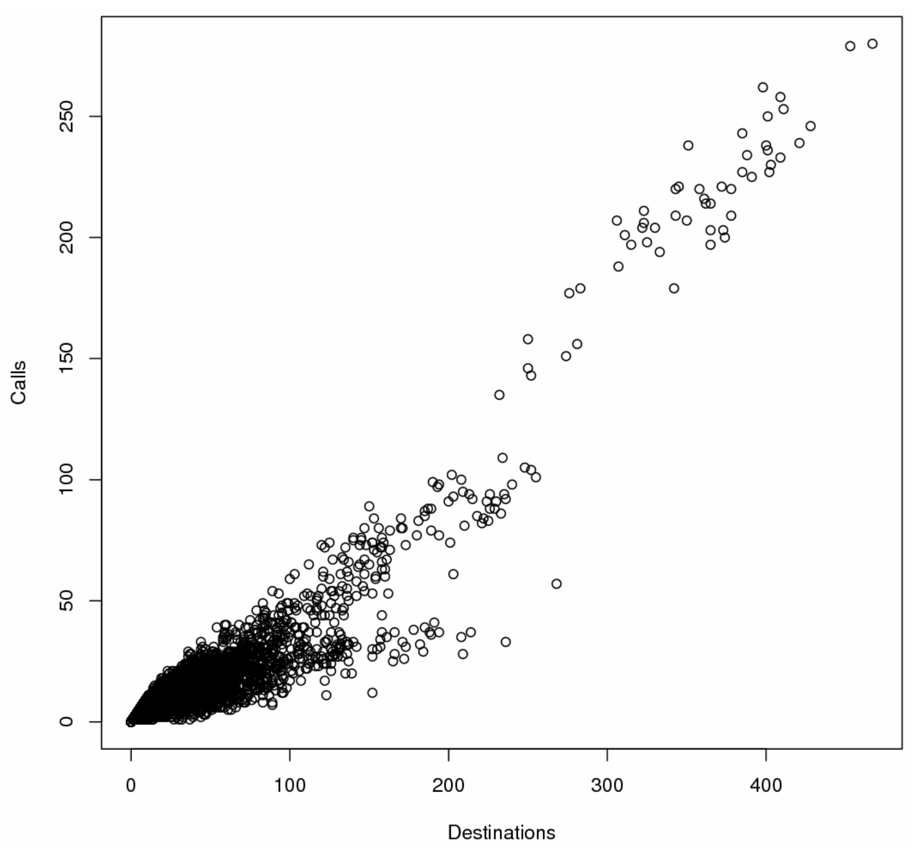


Figure 14. Distribution of destinations for received calls.

Visual inspection reveals a highly suggestive relationship between the count and destination variables, which is supported by a correlation coefficient of 0.880 for send calls, and 0.926 for received calls. A linear regression for calls sent produces

$$D_{sent} = 0.4049C_{sent} + 0.9299$$

and for calls received

$$D_{recv} = 0.4488C_{recv} - 0.2746$$

where D_{sent} and D_{recv} are the dependent variables for sent and received call destinations, and C_{recv} and C_{sent} are the independent variables for counts of sent and received calls.

Given that it is possible to estimate the calls received and sent from the linear model, with a noise factor related to the residual value, the problem, in terms of synthesising representative data, is to distribute the calls over the destination frequency bins. Figure 15 shows the distribution of the average number of calls across 10 destination bins, from the most frequently used to the least frequently used. This takes no account of the number of different destinations.

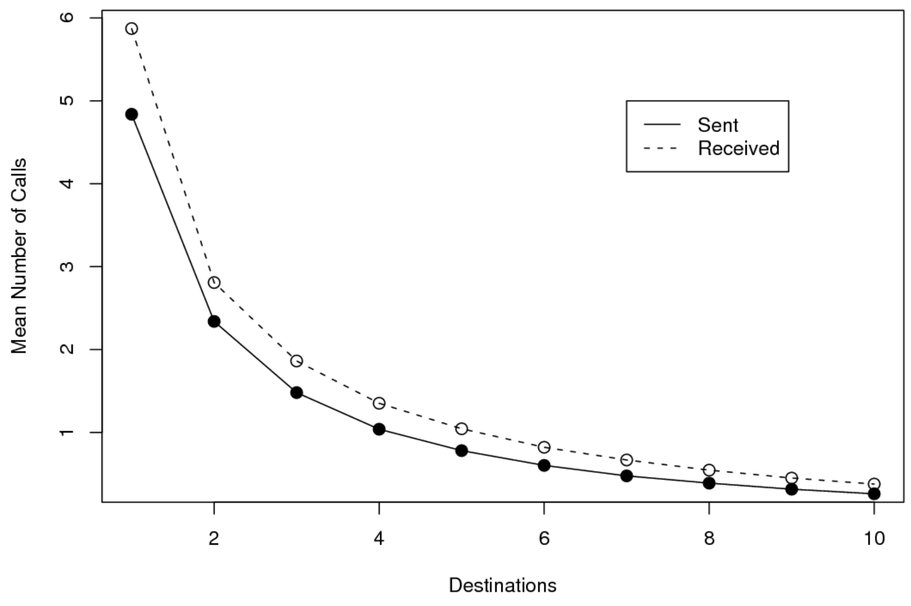


Figure 15. Calls versus destinations.

The average subscriber will receive approximately six calls from the most common destination but send only five calls to the most common destination. Note that these destinations are not necessarily the same. The problem with distributing the call count over the derived destinations is that the above distribution assumes that ten different destinations have been accessed. What is needed is a distribution for each possible value of the destination count greater than one. The appropriate averages can be extracted from the data, as presented in Table 5, with the associated standard deviations in Table 6, where the columns refer to the number of different destinations and the rows give the corresponding values extracted from the raw data set.

10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
6.50	5.81	5.79	5.64	5.02	4.77	4.29	3.96	2.92
3.71	3.12	3.16	2.82	2.52	2.23	1.94	1.75	1.29
2.52	2.19	2.12	1.88	1.64	1.41	1.26	1.11	
1.84	1.62	1.52	1.40	1.25	1.10	1.04		
1.47	1.31	1.22	1.12	1.06	1.01			
1.22	1.12	1.07	1.03	1.00				
1.08	1.04	1.01	1.00					
1.03	1.01	1.00						
1.01	1.00							
1.00								

Table 5. Mean number of calls to different destinations.

10	9	8	7	6	5	4	3	2
4.51	3.89	4.01	4.01	4.09	4.13	3.95	3.95	2.76
2.23	1.68	1.80	1.68	1.70	1.48	1.32	1.29	0.79
1.38	1.09	1.14	1.11	1.02	0.72	0.60	0.40	
0.82	0.80	0.77	0.66	0.64	0.35	0.24		
0.64	0.58	0.49	0.36	0.25	0.14			
0.48	0.34	0.29	0.18	0.07				
0.29	0.20	0.12	0.07					
0.18	0.09	0.04						
0.10	0.05							
0.05								

Table 6. Standard deviations of calls to different destinations.

Note the relatively high values for the standard deviations. For synthesis of representative data, noise should be introduced to the process of call distribution (*e.g.* by assuming that values are normally distributed around the means). Without such noise the destination attributes are determined by a functional relationship derived from the means. For the purpose of distributing actual call numbers across these destinations it is more helpful to have the values normalised, as provided in Table 7.

10	9	8	7	6	5	4	3	2
0.30	0.32	0.34	0.37	0.41	0.46	0.50	0.58	0.69
0.17	0.17	0.19	0.19	0.20	0.21	0.23	0.26	0.31
0.12	0.13	0.13	0.13	0.13	0.13	0.15	0.16	
0.09	0.10	0.09	0.09	0.10	0.10	0.12		
0.07	0.07	0.07	0.08	0.08	0.10			
0.06	0.06	0.06	0.07	0.08				
0.05	0.05	0.06	0.07					
0.05	0.05	0.06						
0.05	0.05							
0.04								

Table 7. Normalised distribution of calls to different destinations.

1.5.3.4 Locations

The treatment of locations follows the same approach as for destinations. Figure 16 and Figure 17 illustrate the number of calls sent and received, respectively, against the number of different locations. Figure 18 shows the distribution of the average number of calls across the 10 location bins, from the most frequently used the least frequently used. This takes no account of the number of different locations.

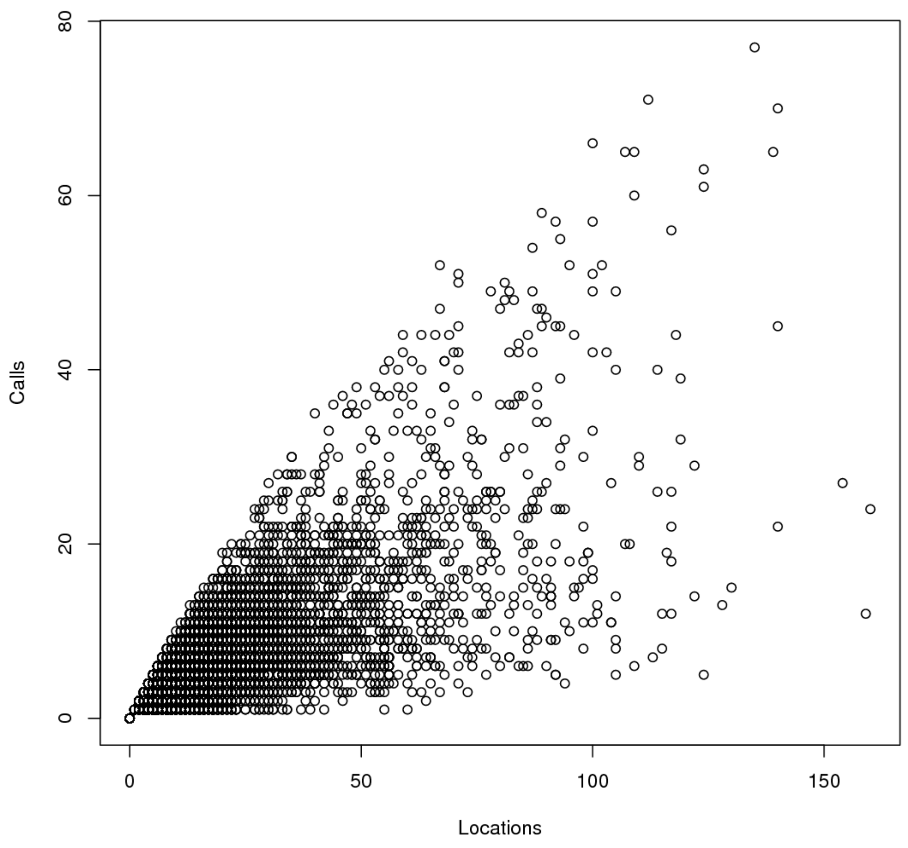


Figure 16. Distribution of locations for sent calls.

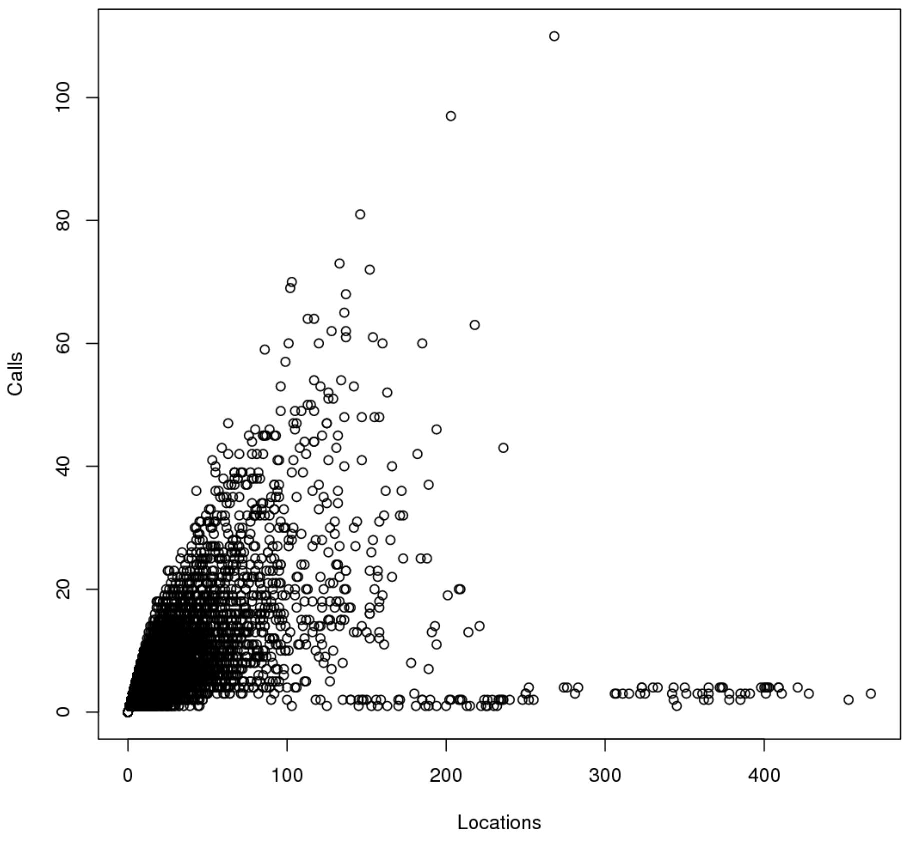


Figure 17. Distribution of locations for received calls.

Visual inspection suggests that there may be a correlation between calls and locations, particularly for received calls, but this is not as convincing as for destinations, with correlation coefficients of 0.771 and 0.492 for sent and received calls, respectively. The plot of received calls suggests some distinct behaviours, which might be indicative of the inclusion of some abnormal calling behaviours in the data set, and thus impact on an otherwise reasonably high correlation.

A linear regression for calls sent produces

$$L_{sent} = 0.2732C_{sent} + 1.3167$$

and for calls received

$$L_{recv} = 0.0966C_{recv} - 3.5633$$

where L_{sent} and L_{recv} are the dependent variables for received and sent call locations, and C_{recv} and C_{sent} are as before.

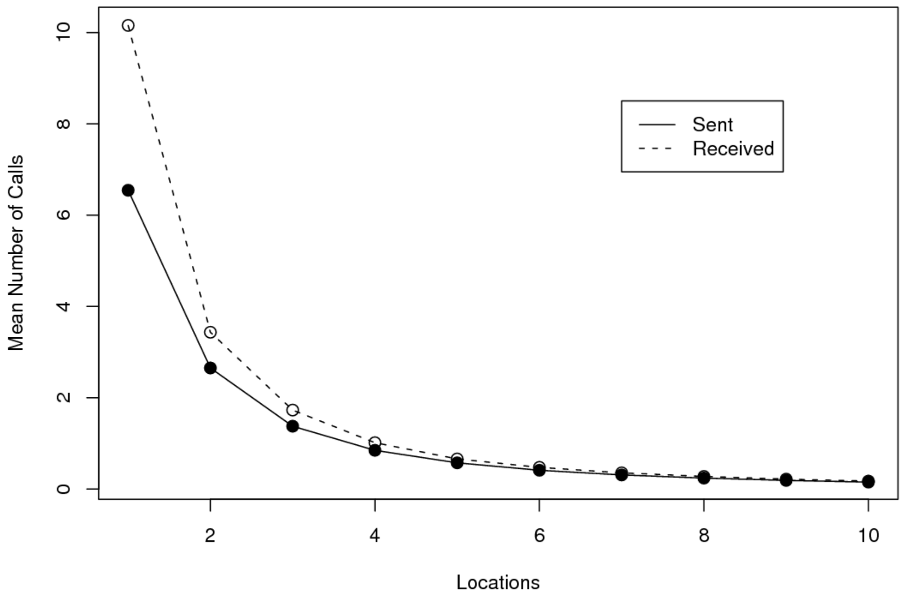


Figure 18. Distribution of calls to locations.

As before, a distribution is needed for each possible value of the location count greater than one. The appropriate averages can be extracted from the data, as presented in Table 8, with the associated standard deviations in Table 9, where the columns refer to the number of different locations and the rows give the corresponding values extracted from the raw data set. Normalised distributions are presented in Table 10.

10	9	8	7	6	5	4	3	2
10.09	10.26	9.41	9.06	8.25	7.58	6.73	5.72	4.51
4.93	4.60	4.71	4.23	3.82	3.32	2.70	2.05	1.51
2.79	2.64	2.51	2.30	2.07	1.80	1.49	1.17	
1.90	1.74	1.72	1.56	1.40	1.25	1.10		
1.40	1.32	1.24	1.24	1.11	1.03			
1.12	1.14	1.07	1.06	1.02				
1.03	1.05	1.02	1.00					
1.01	1.01	1.00						
1.00	1.00							
1.00								

Table 8. Mean number of calls to different locations.

10	9	8	7	6	5	4	3	2
8.87	9.90	8.04	7.98	7.80	7.92	6.51	6.01	4.44
4.38	4.58	5.05	4.24	4.34	3.35	2.79	1.84	1.27
1.68	1.96	1.94	1.94	1.74	1.53	1.03	0.54	
1.21	1.04	1.46	0.99	0.91	0.73	0.42		
0.68	0.65	0.57	0.61	0.42	0.17			
0.37	0.40	0.29	0.27	0.16				
0.17	0.21	0.15	0.06					
0.12	0.11	0.07						
0.00	0.00							
0.00								

Table 9. Standard deviations of calls to different locations.

10	9	8	7	6	5	4	3	2
0.38	0.41	0.41	0.44	0.47	0.51	0.56	0.64	0.75
0.19	0.19	0.21	0.21	0.22	0.22	0.22	0.23	0.25
0.11	0.11	0.11	0.11	0.12	0.12	0.13	0.13	
0.07	0.07	0.08	0.08	0.08	0.08	0.09		
0.05	0.05	0.06	0.06	0.06	0.07			
0.04	0.05	0.05	0.05	0.05				
0.04	0.04	0.04	0.05					
0.04	0.04	0.04						
0.04	0.04							
0.04								

Table 10. Normalised distribution of calls to different locations.

1.5.3.5 Synthesising Call Detail Records

The preceding statistical analysis, of the calling data provided by the operator, provides insight into subscriber behaviour, and also facilitates the synthesis of representative test data. The general process for synthesising feature vectors for training or evaluation is outlined below (where sent and received calls are treated separately).

a. *Total count.* Use the Gamma distributions to generate as many sent and received calls as needed, where the ratio of sent to received should be approximately 57% received and 43% sent.

b. *Call duration.* Use the number of calls sent and the Gamma distributions to generate call duration values.

c. *Distribution over the week*. Using the ratios specified, allocate the call counts to a weekend or a weekday. At this point, noise can be added by distributing the allocation normally around the specified means, ensuring that the total adds up to the number of calls.

d. *Distribute over the day*. Distribute calls over the three daily time periods using the derived ratios, in a similar way to the previous step for allocating to weekend or weekday.

e. *Destinations*. Use the linear relationship to calculate the number of different destinations, introducing appropriate levels of noise. Distribute the call count according to the ratios from the relevant column of the destination table. Again, noise can be introduced, but ensuring that the total call count remains constant.

f. *Locations*. Treat locations in the same way as destinations.

1.6 Prototype module and early findings

This section outlines the findings of work on the prototype system that has informed the design of the AI module. Whilst extensive informal exploratory work has been carried out, the focus here is on examples that demonstrate proof-of-concept.

1.6.1 Data preparation

The data set, as depicted in Figure 4, was divided into three sub-sets (as per the feasibility study reported in Section 1.4.3):

d. Training set. Comprising the first 150 subscribers, representing assumed normal behaviour. The only data used within the evolutionary learning process.

e. Control set. Comprising the next 150 subscribers, representing assumed normal behaviour, used to verify that the detector has been properly trained.

f. Test set. Comprising the final 200 subscribers, represent data with instances of fraud hidden within other seemingly normal data.

The raw CDR file was processed to extract a feature vector for each subscriber for each week of activity. Weeks with zero activity were removed. From the 1024 features, the following feature sub-set was then selected (based on the literatures and research findings) for the results reported herein:

Description	**# attributes**
Descriptive statistics for incoming calls	6
Descriptive statistics for outgoing calls	6
Duration histogram with 8 bins	8
Calls during day, evening, and night slots	3
Total base stations used and histogram of 10 most active	11
Total destinations and histogram of 10 most used	11

Table 11. Selected features.

Values for each attribute were rescaled to the range [0, 1] based on the extreme values of the training set.

1.6.2 Training

The prototype provides an evolutionary learning approach that, for given training data, attempts to develop a detector that shows minimal profile deviations – one training run results in one detector. Figure 19 illustrates how the training process minimises the mean (dimensionless) outputs of the network on the training data (the black trace). As one would hope to see, the control set (supposed normal) shows only marginally higher deviations than one the training set. However, the test set, which contains the hidden fraud cases, shows markedly higher deviations. The training traces show the exponential convergence typically of evolutionary algorithms. In theory, one might expect the difference between training and control sets to begin to diverge at some point, as the neural network effectively memorises the training data. However, this over fitting has not been observed in any training sessions thus far.

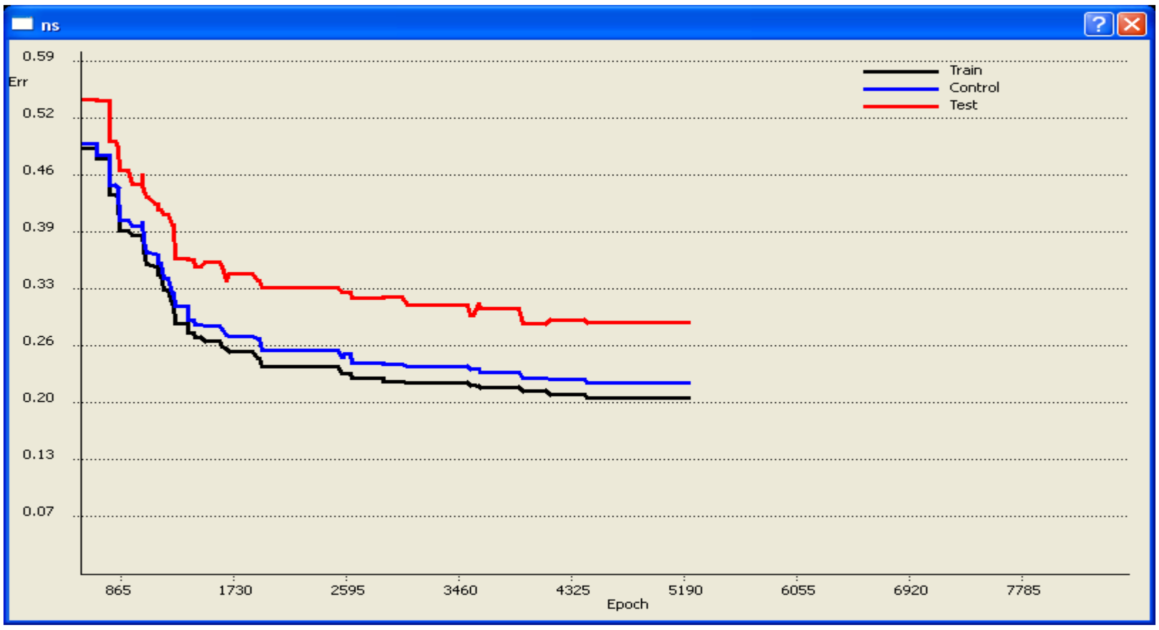


Figure 19. Screenshot from the prototype system, showing profile deviations developing during training.

1.6.3 Misuse indications

In addition to training traces, the prototype also provides the individuals errors per subscriber per week. This is the raw (dimensionless) output of the profiler. Figure 20 shows the output for the sample CDR data (partially trained, as per Figure 19). Important points to note include:

a. The output of the training and control sets are remarkably similar.

b. The peak outputs of the training and control sets are almost identical.

c. The peak output on the training set can be recorded, and a small margin added (e.g. 10%) above which alarms can be generated.

d. The test set shows some extreme deviations from the normal data, giving very clear indications of possible frauds.

e. The output is continuous, and, hence, may be used to prioritise investigations or to generate multiple alarms, each of varying severity.

f. The 30 most anomalous indications have been verified by the operator as examples of fraud.

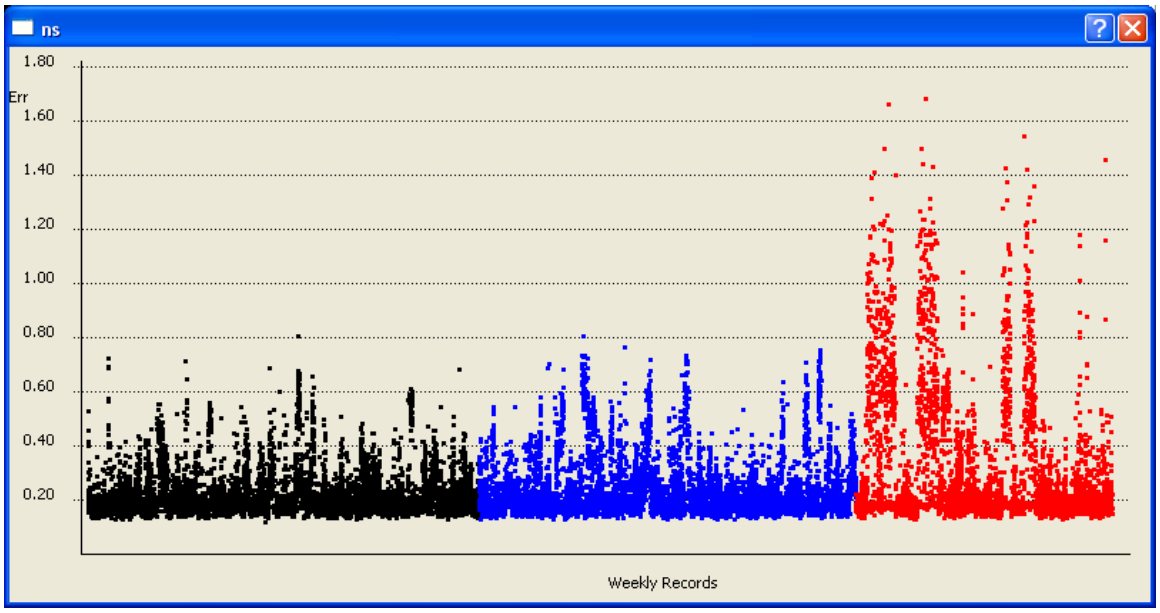


Figure 20. Screenshot from the prototype system, showing deviations developing during training. Only the black trace is used for training. The blue trace represents the control set and the red trace the test set, the latter containing clearly identifiable fraud indications.

1.6.4 Mining

One training cycle results in one trained detector. However, training data may be broken down by various characteristics (*e.g.* call duration, destinations, or mobility), and 'specialised' detectors trained. Each detector has different characteristics, and can be used to identify similar examples from a larger (test) set. Training detectors on different characteristics allows differentiation between subscribers exhibiting fundamentally different calling behaviours. In a sense, it is not critical what the detectors are trained on, as long as they provide different 'views' of the data.

The trained detectors may be readily applied to larger sets of records and the outputs (deviations) from each detector recorded. These raw (dimensionless) outputs may then be analysed to identify groups of subscribers having similar calling behaviours. Such analysis may be accomplished using clustering approaches such as *x*-means. The mining reveals distinct subscriber groups, and, hence, may be of benefit to the operator's business processes, for example, by enabling targeting of services. Additionally, noting the lack of homogeneity of the training data in Figure 20, it is clear that improved differentiation between normal and abnormal behaviour may be achieved by selecting training data from individual clusters. Further, this enables specialised detectors to be deployed, without requiring individual detectors for each subscriber.

The result of mining the supposed normal portion of the CDR samples (using the *Weka* data mining tool for *x*-means clustering), is illustrated in Figure 21. In this case, five clusters have been identified. Profiling, combined with visualisation and clustering tools offers deeper insight into calling patterns than is currently possible, and is a novel contribution to the MDS project.

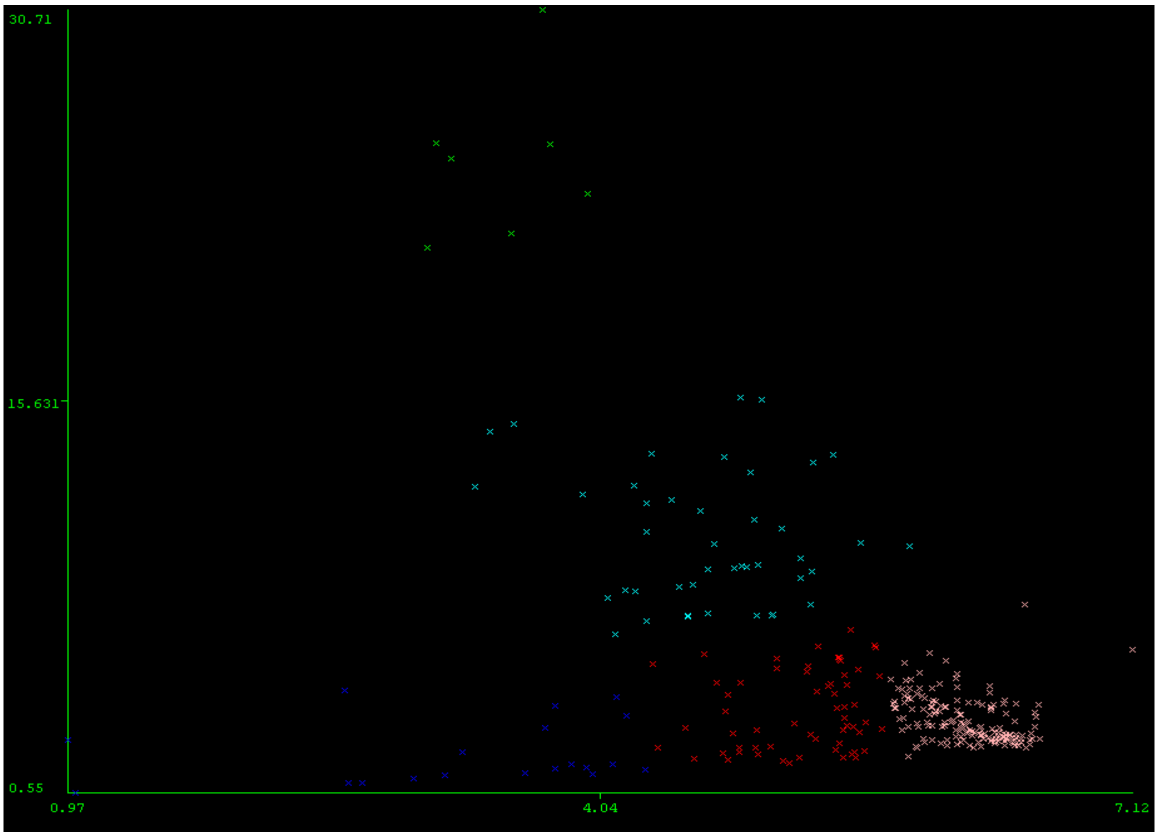


Figure 21. Graphical output from *x*-means clustering on outputs from two detectors trained on different subsets of the data (normal data only). Five clusters are identified, each denoted by a different colour in the plot.

1.6.5 Summary

This section has illustrated some of the research findings from the analysis phase and prototype development. On the data provided by the operator, it has been verified that the selected approach is capable of detecting the class of fraud hidden within this data. The technique is able to profile a given set of CDRs based on summative features, and may thus be adopted for anomaly detection (training on supposed normal behaviour) as well as misuse detection (training on examples on a given misuse class). A novel approach to the mining of CDRs has been developed based on multiple detectors trained on arbitrary but different subsets of samples, with subscriber groups being identified using *x*-means clustering. The use of a neural approach assures that some degree of generalisation is achieved. Hence, the recommendations outlined in Section 1.3.5 have been satisfied.

1.7 System context

This section describes the context of the FD/BP module, and its relationship to other parts of the MDS.

1.7.1 FD/BP module sub-systems

Since the selected AI techniques require training prior to deployment, the FD/BP module requires two subsystems: detector training (DT) and online detection (OD). The data mining process is supported by one or more trained detectors, as well as one or more open source data mining tools (such as *Weka*).

1.7.2 Detector training sub-system

A context diagram for the detector training (DT) sub-system is provided in Figure 22. The purpose of this sub-system is to create a computational machine (referred to as a detector) capable of recognising a usage profile based on examples. Training is based on building a profile of behaviour over a number of defined intervals, where each interval comprises zero or more individual call records for one or more users, selected from a *Historical Data Source*. Input data may be selected, by a domain expert, directly from available CDRs, or may be selected based on information provided by the data mining process. The training process is the same, in either case. The AI engine 'learns' the 'essence' of user behaviour during these periods, and develops a detector that is capable of offering an indication of variation from the learned behaviour. The result of training is a detector that can be deployed to process online data to indicate the extent of variation in behaviours between those present in the training set and those present in the online data. The *Configurator* provides (via a configuration file) values for any configuration parameters, although the module is largely pre-configured prior to delivery. For training, configuration will include the maximum number of training cycles to execute, a target error value, and the desired feedback period. Further configuration parameters may become evident during the implementation phase. The training cycle is controlled via the *Training Interface*, which receives periodic feedback from the sub-system, indicating training cycles executed and current error values. On completion of the training cycle, a trained detector is written to the *Knowledge Repository*.

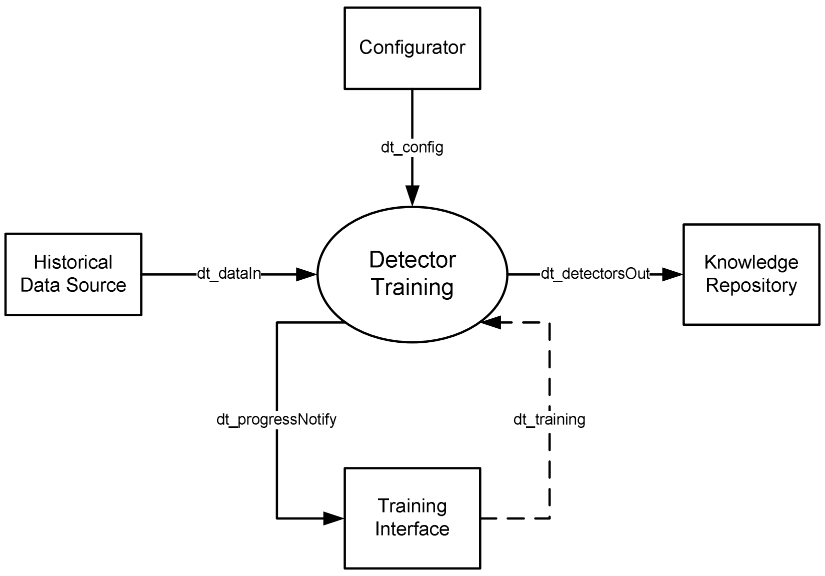


Figure 22. Context for the detector training sub-system.

1.7.3 Online detection sub-system

A context diagram for the online detection (OD) sub-system is provided in Figure 23. This sub-system is concerned with the deployment of one or more previously trained detectors. Trained detectors are loaded from the *Knowledge Repository*. The *Configurator* provides (via a configuration file) values for any configuration parameters. One or more detectors can be loaded and applied to sets of CDRs from the *Online Data Source*. The loaded detectors are capable of raising alarms on the occurrence of either a match or mismatch between the learned behaviours and those present in the online data (*i.e.* anomaly detection or misuse detection). Multiple detectors may be applied simultaneously, to characterise incoming data in different ways, such as for different classes of fraud. Detection is controlled via the *Monitoring Interface*, which then receives any alarm indications or notification of any errors occurring during processing.

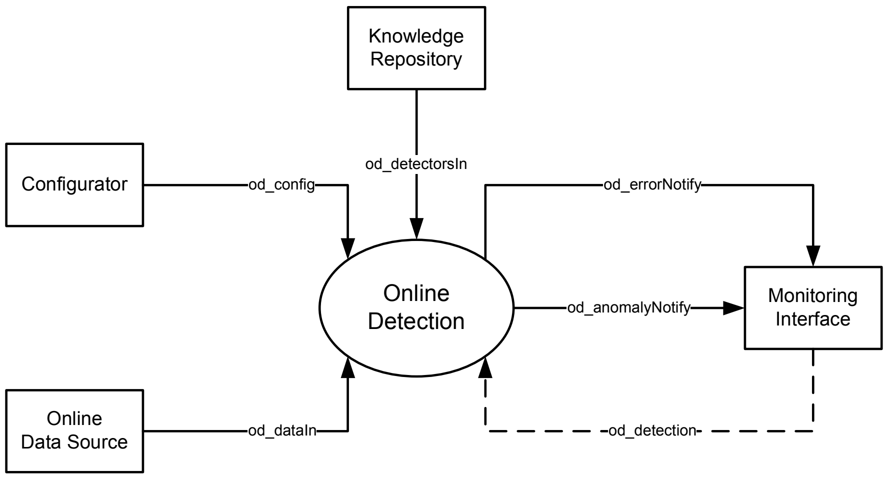


Figure 23. Context for the online detection sub-system.

1.7.4 Data mining

CDRs may be visualised and analysed using a variety of data mining techniques, and thereby identify patterns that exist within the data. These patterns are identified through unsupervised learning techniques (such as cluster analysis) and, therefore, may or may not be of interest in the areas of FD and BP. However, patterns may be found that, when examined by domain experts, can be identified as either behaviours of interest to the operator's business processes or cases of fraudulent usage of the network. Data mining may, therefore, be an end in itself. Additionally, it may be deployed to filter unlabelled data (that is, data of unknown properties) in order to identify cases of interest for learning.

The FD/BP module will not provide bespoke data mining tools. Instead, these will be provided by open source toolsets, such as *Weka*. However, the detector training and online detection sub-systems can be used to facilitate the novel mining approach described in 1.6.4. A number of pre-trained detectors will be provided to support the mining process. Each will be trained on a subset of the sample data provided by the operator, with each subset being selected on the basis of different derived features so that the behavioural patterns learned by each detector are different. The online detection module can then be used to extract measures of deviation across a large data set. A procedure will be specified for clustering these measures, using the *Weka* toolset.

1.7.5 Relationship with other MDS sub-systems

The relationship between the FD/BP AI module and the other MDS sub-systems is depicted in Figure 24. Input CDRs are provided via the mediation layer. Output from the module, in terms of trained detectors, alarms, etc. is sent to the integration framework, with appropriate notifications from there to the DSm. Control and configuration, comes from the DSm, via the integration framework. The knowledge repository, which stores training data, configurations, and trained detectors, is implemented by the integration framework, and managed by the DSm. Translation of AI outputs to the Unified Data Format, is facilitated by the Data Unification Module.

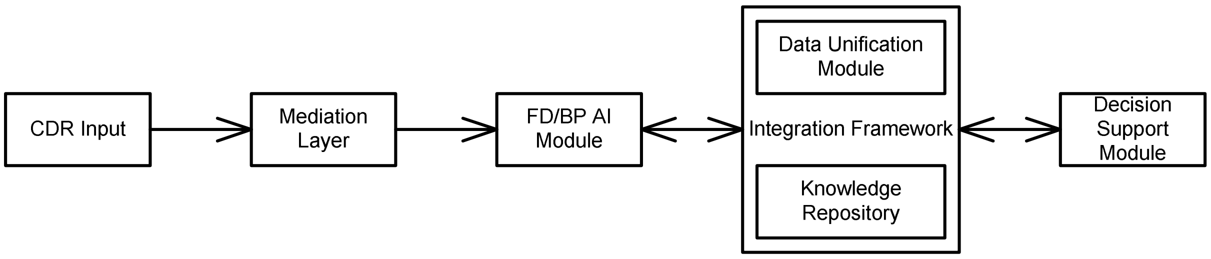


Figure 24. Relationship between the FD/BP AI module and other MDS sub-systems.

1.8 Module architecture

This section describes the specifics of the module design, in sufficient detail to facilitate integration with other components of the MDS. Internal processes, and algorithmic details, are not provided. Results from the prototype system, including verified fraud detection capability, demonstrate that the design of this aspect of the module is satisfactory.

1.8.1 Component view

The detector training component is depicted in Figure 25, which highlights the provided and required interfaces. The detector training sub-system provides the following interfaces:

- *dt_config* for configuring sub-system parameters.
- *dt_dataIn* for establishing source of CDRs for training.
- *dt_detectorOut* for establishing connection to external storage device for writing trained detectors.
- *dt_training* for control of training process.

The detector training sub-system requires the following interfaces:

- *dt_dataInNotify* for accessing CDR records from external storage.
- *dt_detectorOutNotify* for writing trained detectors to external storage device.

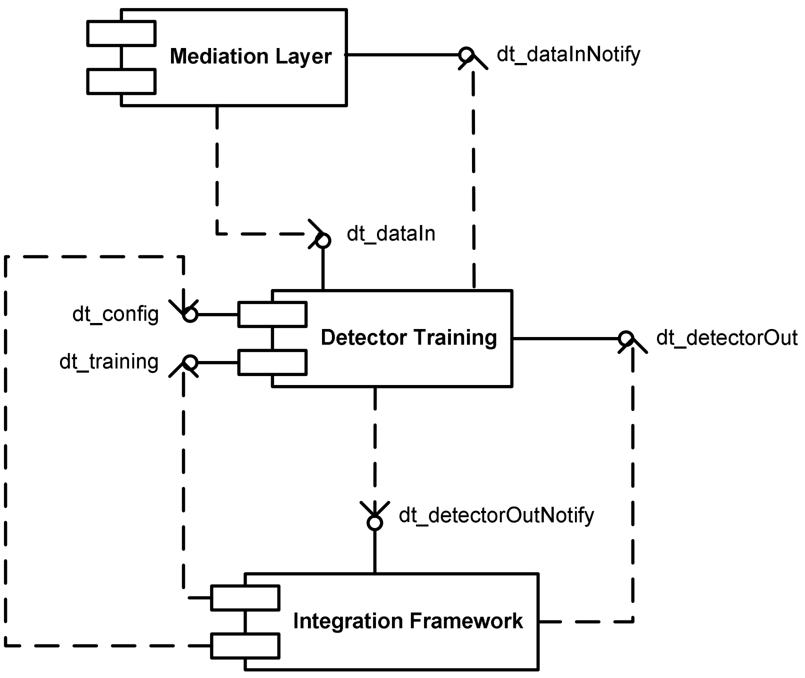


Figure 25. Detector training component.

The online detection component is depicted in Figure 26, which highlights the provided and required interfaces. The online detection sub-system provides the following interfaces:

- *od_config* for configuring sub-system parameters
- *od_dataIn* for input of CDRs for online detection
- *od_detectorIn* for loading and unloading of online detectors from external storage device
- *od_anomaly* for establishing destination for notification of detected anomalies
- *od_detection* for control of online detection process

The online detection sub-system requires the following interfaces:

- *od_anomalyNotify* for receiving anomaly indications
- *od_detectionNotify* for receiving error indications during the detection process

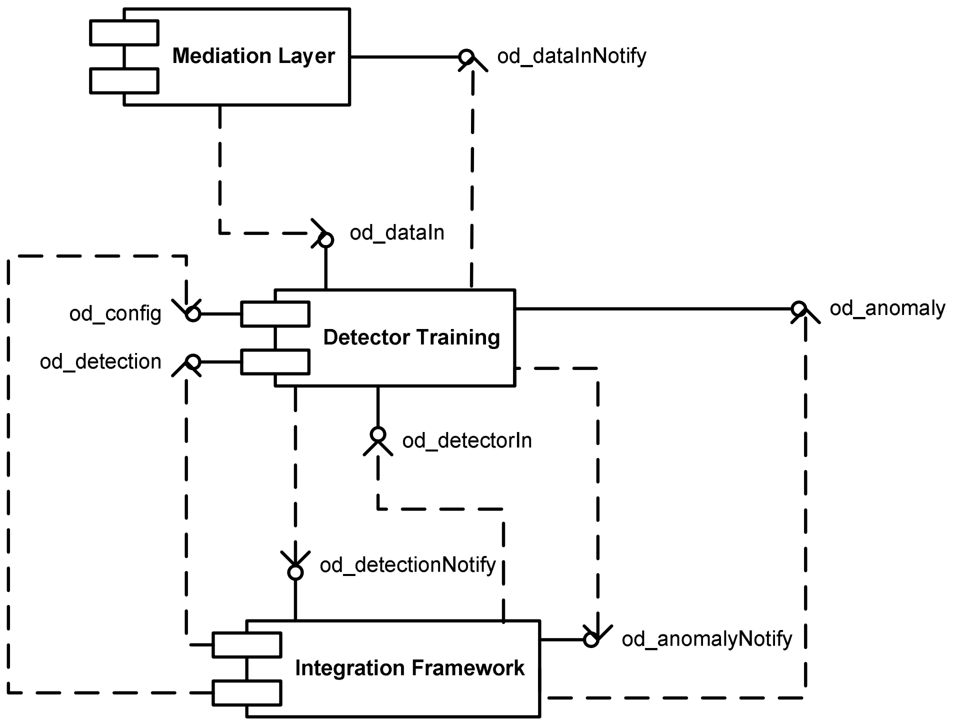


Figure 26. Online detection component.

1.8.2 Application Programming Interface (API)

For performance reasons, the FD/BP module will be implemented in C++, with functionality exposed via a C API. Each sub-system is defined in terms of a number of 'interfaces'. Required interfaces are signified by the use of function pointers. Function prototypes are outlined, below, whilst descriptions are provided within Section 1.8.7. This is a refinement of the specification provided in the requirements document (D2.4.1).

```
/* defined types */

typedef struct
{
   int day;
   int month;
   int year;
}TDate;

typedef struct
{
   int hour;
   int minute;
   int second;
}TTime;

typedef struct
{
   char *pPhoneNumber1;
   char *pPhoneNumber2;
   char *pPhoneNumber3;
   char *pMobileType;
   char *pIPSUserLabel;
   int   direction;          // 0 = to network, 1 = from network, 2 =
forwarded
   int   completionStatus;   // 0 = completed, 1 = dropped
   TDate date;
   TTime time;
   int   duration;
}TCallRecord;

typedef enum
{
   rc_success = 0,
   // other values to be defined
}TReturnCode;

/* interface dt_config */

TReturnCode dt_config_loadConfig    (char *path);

void dt_config_defaultConfig (void);

/* interface dt_dataIn */

void dt_dataIn_setDataInput (
   int          (*dt_dataInNotify_getNumPeriods) (void                   ),
   int          (*dt_dataInNotify_getNumRecords) (int period             ),
   TCallRecord *(*dt_dataInNotify_getCallRecord) (int period,  int record),
   void         (*dt_dataInNotify_free         ) (TCallRecord *callRecord)
                            );

/* interface dt_detectorOut */

void dt_detectorOut_setWriter (
   void (*dt_detectorOutNotify_write) (char *pData, int size, double error)
                              );

/* interface dt_training */
```

```
TReturnCode dt_training_run (
   void (*dt_trainingNotify_progress) (int epoch, double error)
                            );

void dt_training_stop (void);

/* interface od_config */

TReturnCode od_config_loadConfig (char *path);

void od_config_defaultConfig (void);

/* interface od_dataIn */

void od_dataIn_setDataInput (
   int          (*od_dataInNotify_getNumRecords) (void                   ),
   TCallRecord *(*od_dataInNotify_getCallRecord) (int record             ),
   void         (*od_dataInNotify_nextPeriod   ) (void                   ),
   void         (*od_dataInNotify_free         ) (TCallRecord *callRecord)
                            );

/* interface od_detectorIn */

TReturnCode od_detectorIn_loadDetector (char   *pDetector,
                                        int    size,
                                        int    uid,
                                        int    detectionType,
                                        double threshold);

TReturnCode od_detectorIn_unloadDetector (uid);

/* interface od_anomaly */

void od_anomaly_setAnomalyNotify (
   void (*od_anomalyNotify_anomaly) (int     uid,
                                     int     detectionType,
                                     double  threshold,
                                     double  actual,
                                     char   *subscriber)
                                 );

/* interface od_detection */

TReturnCode od_detection_run (
   void (*od_detectionNotify_error) (TReturnCode error)
                             );

void od_detection_stop (void);
```

1.8.3 Detector training component

The principal elements of the detector training component are illustrated in Figure 27. The responsibilities of the individual classes are defined, below.

a. *TrainingManager*. Responsible for the configuration of the evolutionary learning engine and the control of the training process. This class interacts with the integration framework via the *dt_config* and *dt_training* interfaces.

b. *EvolutionaryLearningEngine*. Responsible for training detectors to profile the features extracted from the input CDRs. Implements an evolutionary algorithm in order to create neural network based detectors. Interacts with the integration framework via the *dt_detectorOut* interface.

c. *Detector*. Used to evaluate various combinations of neural network parameters developed by the evolutionary learning engine.

d. *DataReader*. Reads raw CDR data provided by the mediation layer, via the *dt_dataIn* interface. CDRs are read into the *CDRBuffer* in preparation for feature extraction.

e. *CDRBuffer*. Stores CDRs in preparation for feature extraction. Local storage of CDRs is necessary due to the multi-pass nature of the feature extraction process.

f. *FeatureExtractor*. Extracts features from the raw CDR data, generating one feature vector for a given set of CDRs. Feature vectors are stored in the *FeatureBuffer* in preparation for training.

g. *FeatureBuffer*. Stores extracted feature vectors to be used by the evolutionary learning engine for developing neural network based detectors. Local storage of extract features is necessary due to the multi-pass nature of the learning process.

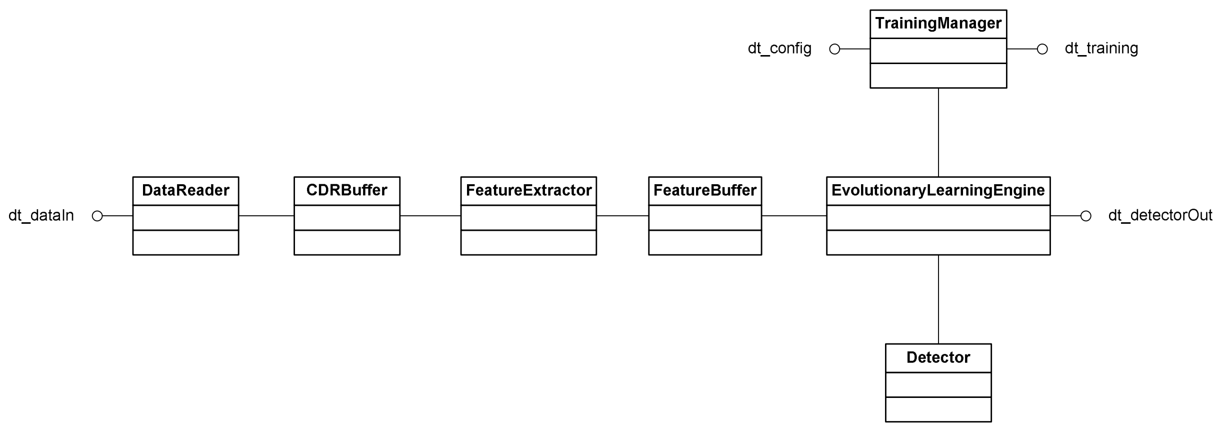


Figure 27. Decomposition of the detector training component.

1.8.4 Online detection component

The principal elements of the online detection component are illustrated in Figure 28. The responsibilities of the individual classes are defined, below.

a. *DetectionManager*. Responsible for the configuration of the execution engine and the control of the detection process. This class interacts with the integration framework via the *od_config* and *od_detection* interfaces.

b. *ExecutionEngine*. Responsible for delivering extracted features to loaded detectors, and the thresholding of detector outputs in order to generate anomaly indications. Interacts with the integration framework via the *dt_anomaly*.

c. *DetectorManager*. Manages the list of loaded detectors. Interacts with the integration framework via the *od_detectorIn* interface, through which detectors may be loaded and unloaded.

d. *Detector*. Pre-trained neural network based profilers used to measure deviations between present features and those from the training data.

e. *DataReader*. Reads raw CDR data provided by the mediation layer, via the *dt_dataIn* interface. CDRs are read into the *CDRBuffer* in preparation for feature extraction.

f. *CDRBuffer*. Stores CDRs in preparation for feature extraction. Local storage of CDRs is necessary due to the multi-pass nature of the feature extraction process.

g. *FeatureExtractor*. Extracts features from the raw CDR data, generating one feature vector for a given set of CDRs. Feature vectors are passed to the execution engine for anomaly/misue detection.

h. *FeatureBuffer*. Stores extracted feature vectors to be used by the evolutionary learning engine for developing neural network based detectors. Local storage of extract features is necessary due to the multi-pass nature of the learning process.

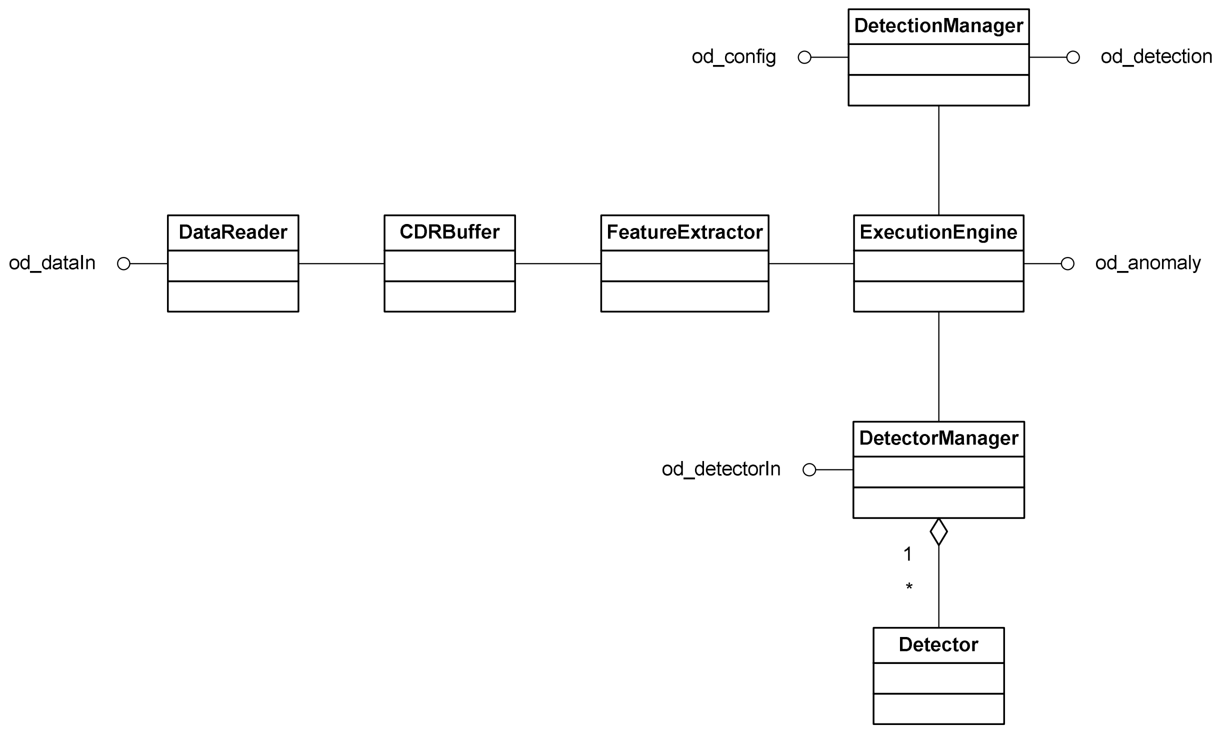


Figure 28. Decomposition of the online detection component.

1.8.5 Behavioural models

This section describes the behaviour of the FD/BP sub-systems using Role Activity Diagrams (RADs). This represents effectively a refinement of the RADs from the requirements document, D2.4.1, to reflect design level concerns. For example, certain actions depicted within the roles at requirements time are implemented as interactions between specific component parts of the system (that is, internal system issues are now considered). During requirements this is not of interest; that is, the user is only concerned that such an action is possible, not how it comes about (*e.g.* clicking a button on the graphical user interface). At this stage, however, such interactions must be defined. A RAD offers a more flexible notation for such a model (being developed, originally, from Petri-nets), which cannot be easily represented using the behavioural models of the UML, such as statecharts, sequence diagrams, or activity diagrams. As the RAD notation may not be as familiar as those notations of the UML, Figure 29 provides an overview of the main constructs. These constructs are combined in a graph that represents effectively a state machine, although information flow can be readily reflected. The behavioural model is further enhanced by the description of 'typical' usage scenarios, in Section 1.8.6.

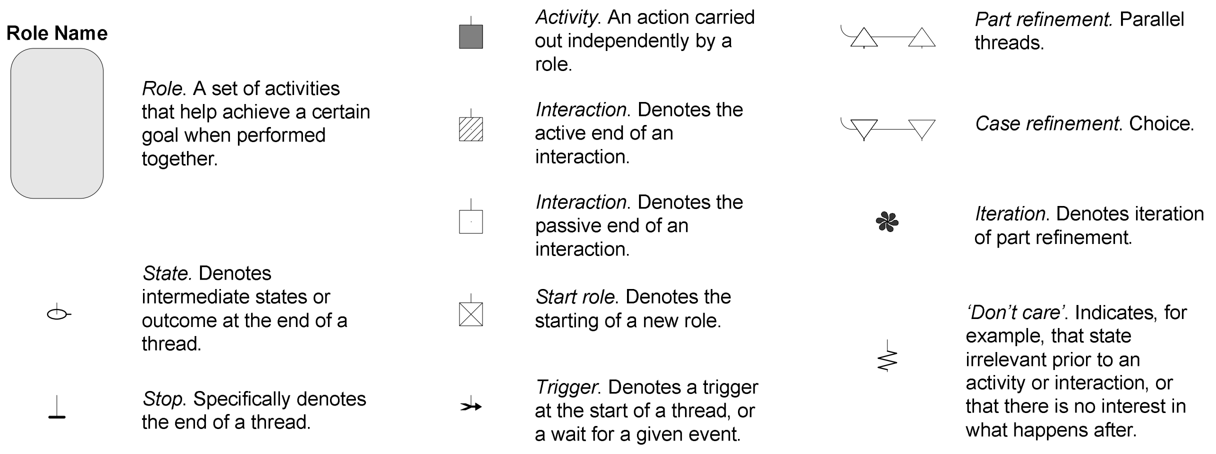

Role Name
Role. A set of activities that help achieve a certain goal when performed together.
State. Denotes intermediate states or outcome at the end of a thread.
Stop. Specifically denotes the end of a thread.
Activity. An action carried out independently by a role.
Interaction. Denotes the active end of an interaction.
Interaction. Denotes the passive end of an interaction.
Start role. Denotes the starting of a new role.
Trigger. Denotes a trigger at the start of a thread, or a wait for a given event.
Part refinement. Parallel threads.
Case refinement. Choice.
Iteration. Denotes iteration of part refinement.
'Don't care'. Indicates, for example, that state irrelevant prior to an activity or interaction, or that there is no interest in what happens after.

Figure 29. Overview of notational elements of a Role Activity Diagram (RAD).

The RADs highlight the internal behaviour of the sub-systems, in response to interactions with other parts of the MDS. It is not the intention to impose particular models on other parts of the MDS, however, the models do describe the correct form of interaction that the knowledge repository, mediation layer, decision support module (DSm), must supply in order to function properly with the AI modules. These can, therefore, be seen as necessary provisions, if not sufficient or complete. The following roles are modelled in both cases:

a. Detector training / online detection sub-system. Encompasses all of the behaviour of the AI module.

b. Decision support module (via integration framework). Although the communications with the AI module are facilitated entirely by the integration framework, the behaviour is determined solely by the logic of the decision support module.

c. Knowledge repository (integration framework). This role encapsulates the external (to the AI module) storage of, for example, trained detectors, training data, and other data required for running and maintaining the system. In practice, this module is an integral part of the integration framework.

d. Mediation layer. The mediation layer supplies the input data (CDRs), although this may be via the integration framework.

1.8.5.6 Detector training

The RAD focussing on the detector training (DT) sub-system in shown in Figure 30. In this model, the mediation layer, as the input source for CDRs, and the knowledge repository, as the destination for trained detectors, are simply shown without particular regard to their internal models. The remainder of the model is devoted to the DT sub-system and the behaviours that its proper usage will imply on the DSm. Note that the DT sub-system is a passive role until training has started, when it becomes active. Similarly, the DSm role is active until training is started, when it then plays a largely passive role (except for the ability to stop training) in processing feedback from the DT sub-system.

The upper part of both the DT and DSm models, encompassing the activities from the *initial* state until the *ready* is reached, are concerned with configuration. The structures within the DSm reflect the requirements of the DT API in terms of mandatory or optional configuration activities, and the order in which they may be performed. Since the structures are identical, here, we only describe the DT model.

From the *initial* state there are three threads, signifying that the order in which configuration occurs is not relevant. The first thread concerns the loading of configuration files and/or restoration of the default configuration. There are three choices: to read and process a configuration file, to restore the default configuration, or take no action. Hence, this thread denotes the optional aspects of the configuration of the DT sub-system. The choice is defined by the interaction between the DSm (the active party) and DT sub-system (the passive party). The second thread concerns establishing the connection to the source of CDRs, whilst the third thread concerns establishing the connection to the destination for trained detectors. Both are mandatory. Only once all three threads have completed (states *configured*, *connected to input*, and *connected to output*), does the system move to the *ready* state, from which training may commence.

Training is initiated by the DSm, with the *start training* interaction. This results in two threads, in both the DSm and DT roles. The first caters for the possibility that the user might wish to stop training before the activity is complete. The *stop training* interaction, driven by the DSm, results in the *stop requested* state within the DT sub-system. The second thread within the

DT sub-system is actively engaged in training cycles. At the start of each training cycle, this thread checks the state to see if a stop has been received and if so, returns to the *initial* state. If not, then the thread proceeds to either *send progress notifications* to the DSm, at intervals determined by the selected configuration, or process the input data and train the detector. Data is read from the *mediation layer* under the control of the DT sub-system. On completion of the training cycle, the detector is written to the *knowledge repository* and the DT sub-system returns to the initial state. Once training is underway, one thread in the DSm responds to progress notifications from the DT sub-system. The response to such notifications is a concern for the DSm design, so not defined here.

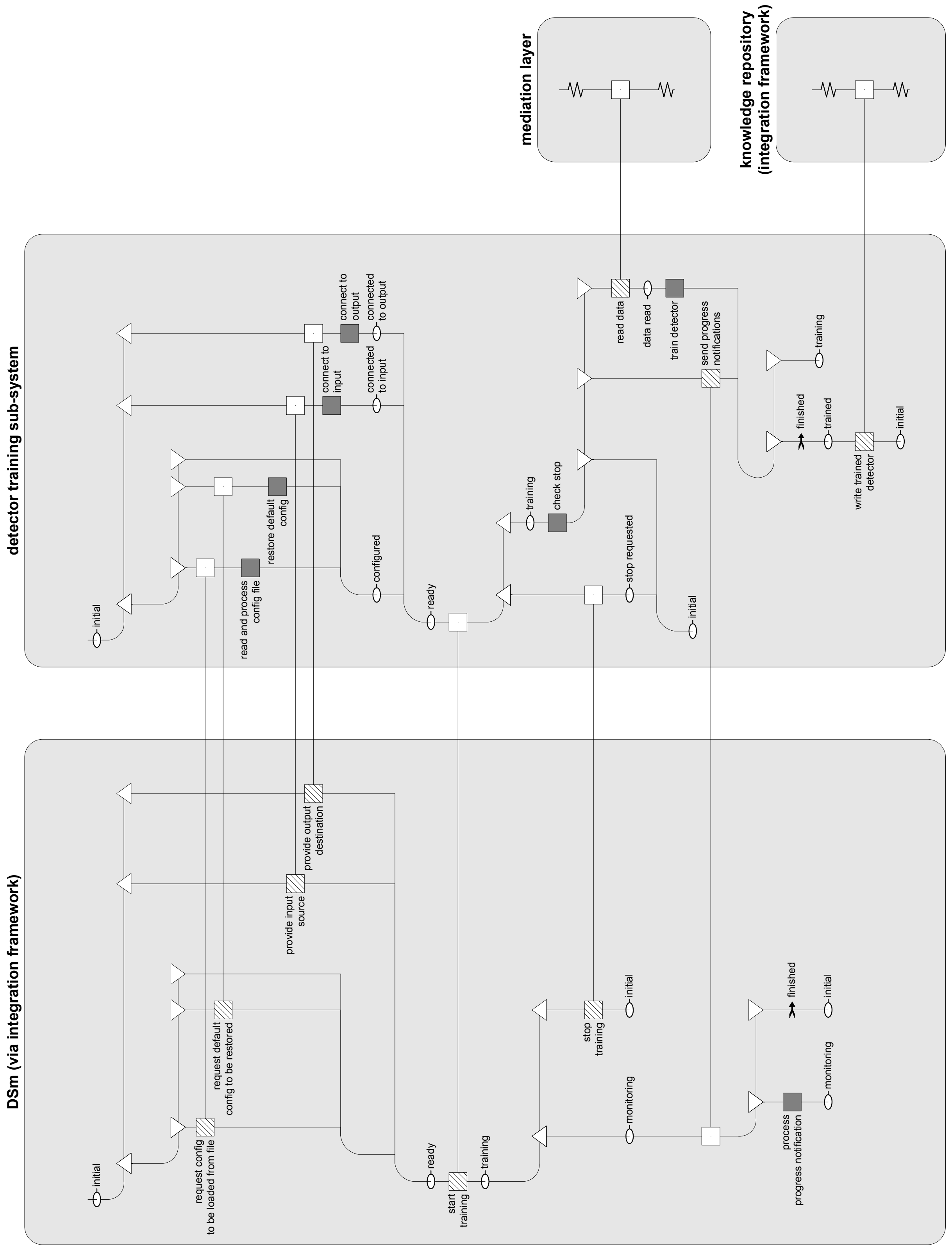


Figure 30. Role Activity Diagram for the detector training sub-system.

1.8.5.7 Online detection

The RAD focussing on the online detection (OD) sub-system in shown in Figure 31. In this model, the mediation layer, as the input source for CDRs, and the knowledge repository, as the source of trained detectors, are simply shown without particular regard to their internal models. The remainder of the model is devoted to the OD sub-system and the behaviours that its proper usage will imply on the DSm. Note that the OD sub-system is a passive role until detection has started, when it becomes active. Similarly, the DSm role is active until detection is started, when is then plays a somewhat mixed role, driving interactions related to the loading and unload of detectors, and stopping detection, but is otherwise responding to notifications from the OD sub-system.

The upper part of both the OD and DSm models, encompassing the activities from the *initial* state until the *ready* is reached, are concerned with configuration. The structures within the DSm reflect the requirements of the OD API in terms of mandatory or optional configuration activities, and the order in which they may be performed. Since the structures are identical, here, we only describe the OD model.

From the *initial* state there are three threads, signifying that the order in which configuration occurs is not relevant. The first thread concerns the loading of configuration files and/or restoration of the default configuration. There are three choices: to read and process a configuration file, to restore the default configuration, or take no action. Hence, this thread denotes the optional aspects of the configuration of the OD sub-system. The choice is defined by the interaction between the DSm (the active party) and OD sub-system (the passive party). The second thread concerns establishing the connection to the source of CDRs, whilst the third thread concerns establishing the connection to the destination for trained detectors. Both are mandatory. Only once all three threads have completed (states *configured*, *connected to input*, and *connected to output*), does the system move to the *ready* state, from which detection may commence.

In separate threads, in both the DSm and OD sub-systems, provision is made for the loading and unloading of previously trained detectors. The related thread within the DSm offers two choices, for either loading a detector or unloading a previously loaded detector. Loading involves an interaction with the *knowledge repository*, where trained detectors are stored. This, in turn, results in an interaction between the *knowledge repository* and the OD sub-system. Unloading involves an interaction between the DSm and the OD sub-system, directly. The *knowledge repository* is otherwise passive. Two corresponding threads in the OD sub-system support loading and unloading respectively. The OD sub-system maintains a collection of loaded detectors, and the special cases of empty and loaded (*i.e.* one or more detectors are loaded) are handled, in particular because detection will not proceed if no detectors are loaded. Otherwise, the loading and unloading can be done on-the-fly.

Detection is started by the DSm, with the *start detection* interaction, which moves this main thread to the *detecting* state. From here, three threads handle the various interactions with the OD sub-system. The first two deal with notifications of either anomaly events or errors occurring during the process of the current batch of CDRs. The third allows the DSm to request the OD sub-system to *stop detecting*.

Once detecting has been started by the DSm, the OD sub-system has two threads. The first caters for the possibility that the user might wish to stop the detection process. The *stop detecting* interaction, driven by the DSm, results in the *stop requested* state within the OD sub-system. The second thread within the OD sub-system is actively engaged in detection cycles. At the start of each detection cycle, this thread checks the state to see if a stop has been received and if so, returns to the *initial* state. If not, then the thread proceeds to check whether any detectors are loaded (the *loaded* state). If not, the thread returns to the *detecting* state. If there are detectors loaded, data is read by interacting with the mediation layer, and processed by running the detectors. At this point, there are three choices: if an anomaly is detection, it is communicated to the DSm; if an error occurs during processing, it is communicated to the DSm; if no anomalies are detection and no errors occur, the thread returns to the detecting state. The processing of anomalies and errors is a concern for the DSm design, and therefore, is not defined here.

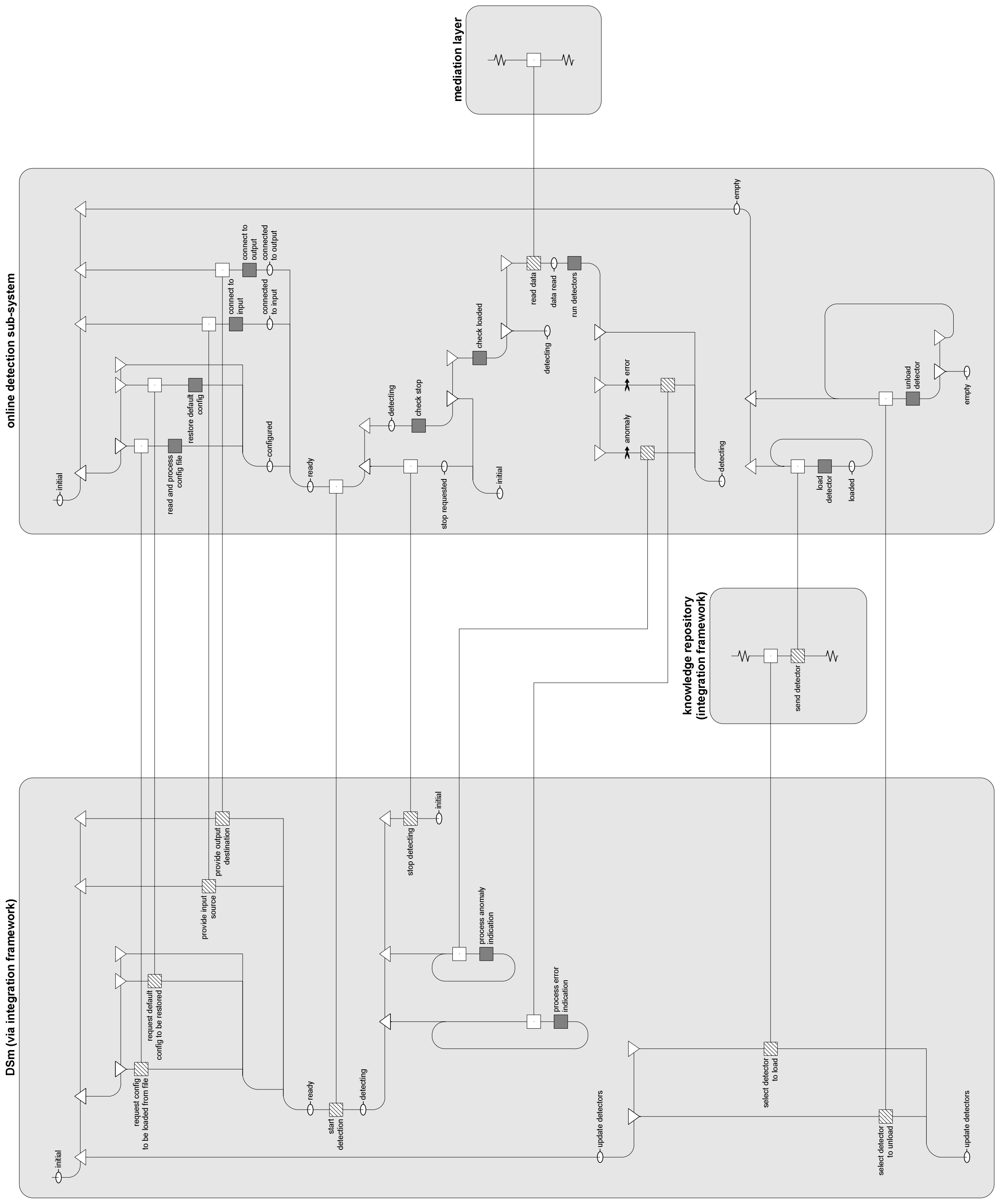


Figure 31. Role Activity Diagram for the online detection sub-system.

1.8.6 Usage scenarios

This section clarifies the usage of the FD/BP AI module, highlighting the typically steps involved in preparing and executing training and detection cycles.

1.8.6.8 Training

Steps 1 – 3 can be performed in any order. Step 1 is optional, steps 2 and 3 are obligatory, prior to training.

1. Load configuration, restore the default configuration, or leave the configuration alone (either previously loaded configuration or default configuration). This step is optional.

```
TReturnCode dt_config_loadConfig (char *path);
void dt_config_defaultConfig (void);
```

2. Connect the training sub-system to a source of input data (provided by the mediation layer). Data will be read under the training sub-system's control, hence, connecting the data source essentially involves providing handles to four callback functions.

```
void dt_dataIn_setDataInput (
   int          (*dt_dataInNotify_getNumPeriods) (void                  ),
   int          (*dt_dataInNotify_getNumRecords) (int period            ),
   TCallRecord *(*dt_dataInNotify_getCallRecord) (int period, int record ),
   void         (*dt_dataInNotify_free         ) (TCallRecord *callRecord)
                           );
```

3. Provide a destination for the trained detectors. Detectors are written to a storage device external to the training sub-system (provided by the integration framework) at the end of the training cycle.

```
void dt_detectorOut_setWriter (
   void (*dt_detectorOutNotify_write) (char *pData, int size, double error)
                             );
```

4. Initiate the training process. Optionally, provide a callback function for periodic progress notifications. Training can be stopped if needed.

```
TReturnCode dt_training_run (
   void (*dt_trainingNotify_progress) (int epoch, double error)
                           );
void dt_training_stop (void);
```

1.8.6.9 Online Detection

Online detection is similar in terms of set up. Step 1 is optional. Steps 2 and 3 are obligatory, prior to detection.

1. Load configuration, as per training. This step is optional.

```
TReturnCode od_config_loadConfig (char *path);
void od_config_defaultConfig (void);
```

2. Connect to input source. This 'online' data is provided by the mediation layer, to be read under the control of the detection sub-system.

```
void od_dataIn_setDataInput (
   int          (*od_dataInNotify_getNumRecords) (void                  ),
   TCallRecord *(*od_dataInNotify_getCallRecord) (int record            ),
   void         (*od_dataInNotify_nextPeriod   ) (void                 ),
   void         (*od_dataInNotify_free         ) (TCallRecord *callRecord)
                           );
```

3. Set destination for anomaly indications. These are sent as and when discovered by the detection sub-system.

```
void od_anomaly_setAnomalyNotify (
   void (*anomalyNotify_anomaly) (int     uid,
                                  int     detectionType,
                                  double  threshold,
                                  double  actual,
                                  char   *subscriber)
                                 );
```

4. Load one or more detectors into the detection sub-system, from external storage device provided by the integration framework, as previously stored by the training sub-system. Detectors can be loaded/unloaded on-the-fly.

```
TReturnCode od_detectorIn_loadDetector (char  *pDetector,
                                        int    size,
                                        int    uid,
                                        int    detectionType,
                                        double threshold);
TReturnCode od_detectorIn_unloadDetector (uid);
```

5. Run detection sub-system. Optionally, provide a callback function for notification of any errors that occur during processing. Training can be stopped if needed.

```
TReturnCode od_detection_run (
   void (*od_detectionNotify_error) (TReturnCode error)
                                   );
void od_detection_stop (void);
```

1.8.7 Function descriptions

Function descriptions for the API described in Section 1.8.2, are provided below.

Interface	`dt_config`
Method name	`dt_config_loadConfig`
Purpose	To load a configuration file for the detector training sub-system.
Parameters	
`path`	Null terminated string defining full path to configuration file.
Returns	Return code indicating success/failure.
Preconditions	Detector training sub-system is stopped.
Notes	

Interface	`dt_config`
Method name	`dt_config_defaultConfig`
Purpose	To restore the default configuration of the detector training sub-system.
Parameters	
Returns	
Preconditions	Detector training sub-system is stopped.

Notes	Default values determined during implementation.

Interface	`dt_dataIn`
Method name	`dt_dataIn_setDataInput`
Purpose	To set callback functions for reading call records from external storage device.
Parameters	
`dt_dataInNotify_getNumPeriods`	Function pointer. See separate callback contract.
`dt_dataInNotify_getNumRecords`	Function pointer. See separate callback contract.
`dt_dataInNotify_getCallRecord`	Function pointer. See separate callback contract.
`dt_dataInNotify_free`	Function pointer. See separate callback contract.
Returns	
Preconditions	Detector training sub-system is stopped.
Notes	

Interface	`dt_dataInNotify`
Method name	`dt_dataInNotify_getNumPeriods`
Purpose	Callback function to allow detector training sub-system to ascertain the number of periods of data in the training set to be profiled.
Parameters	
Returns	Number of periods of data in the training set.
Preconditions	
Notes	

Interface	`dt_dataInNotify`
Method name	`dt_dataInNotify_getNumRecords`
Purpose	Callback function to allow detector training sub-system to ascertain the number of records within the specified period of training data.
Parameters	
`period`	Index of period to query.
Returns	Number of records within period.
Preconditions	
Notes	

Interface	`dt_dataInNotify`
Method name	`dt_dataInNotify_getCallRecord`
Purpose	Callback function to allow the detector training sub-system to acquire a handle to the specified call record from external storage device.
Parameters	
`period`	Index of period.
`record`	Index of record.
Returns	Handle to call record.
Preconditions	
Notes	See also `dt_dataInNotify_free`.

Interface	`dt_dataInNotify`
Method name	`dt_dataInNotify_free`
Purpose	Callback function to allow the detector training sub-system to inform external storage device that it has finished with a handle to a call record. Facilitates memory management; may be stubbed if not required.
Parameters	
`callRecord`	Handle to record to be freed.
Returns	
Preconditions	Record has been previously acquired (see `dt_dataInNotify_getCallRecord`).

Notes	

Interface	`dt_detectorOut`	
Method name	`dt_detectorOut_setWriter`	
Purpose	To set callback function to be used by the detector training sub-system for writing trained detector(s) to external storage device.	
Parameters		
`dt_detectorOutNotify_write`	Function pointer. See separate callback contract.	
Returns		
Preconditions	Detector training sub-system is stopped.	
Notes		

Interface	`dt_detectorOutNotify`
Method name	`dt_detectorOutNotify_write`
Purpose	To write a trained detector to external storage device.
Parameters	
`pData`	Handle to memory location containing packed detector to be written.
`size`	Number of bytes to write.
`error`	Measure of the error of the trained detector on the training data.
Returns	
Preconditions	
Notes	Error indicator may be used by training expert to determine whether detector should be retained, or by analyst to determine whether to deploy the detector and a suitable threshold to apply (see `od_detectorIn_loadDetector`).

Interface	`dt_training`
Method name	`dt_training_run`
Purpose	To initiate a training cycle.
Parameters	
`dt_trainingNotify_progress`	Function pointer. See separate callback contract.
Returns	Return code indicating success/failure.
Preconditions	Input source and detector output connected (configuration is optional). Detector training sub-system is stopped.
Notes	Callback function `dt_trainingNotify_progress` may be set to `null`, in which case it is ignored. Otherwise, this function will be called periodically to indicated progress on the training cycle.

Interface	`dt_training`
Method name	`dt_trainingNotify_progress`
Purpose	Hook to acquire progress indication from the training cycle.
Parameters	
`epoch`	Indicator of progress – current training epoch.
`error`	Indicator of progress – measure of the error of the detector on the training data.
Returns	
Preconditions	
Notes	Interpretation of progress indicators to be determined at design time.

Interface	`dt_training`
Method name	`dt_training_stop`
Purpose	To interrupt the training process.
Parameters	
Returns	
Preconditions	Detector training sub-system is started.

Notes	If a training cycle has been initiated, all progress will be lost, and the sub-system restored to a state ready for training to begin. There is no guarantee how quickly the sub-system will respond – this call will block until the sub-system is ready.

Interface	`od_config`
Method name	`od_config_loadConfig`
Purpose	To load a configuration file for the online detection sub-system.
Parameters	
`path`	Null terminated string defining full path to configuration file.
Returns	Return code indicating success/failure.
Preconditions	Online detection sub-system is stopped.
Notes	

Interface	`od_config`
Method name	`od_config_defaultConfig`
Purpose	To restore the default configuration of the online detection sub-system.
Parameters	
Returns	
Preconditions	Online detection sub-system is stopped.
Notes	Default values determined at design time.

Interface	`od_dataIn`
Method name	`od_dataIn_setDataInput`
Purpose	To set callback functions for reading call records from online data stream.
Parameters	
`od_dataInNotify_getNumRecords`	Function pointer. See separate callback contract.
`od_dataInNotify_getCallRecord`	Function pointer. See separate callback contract.
`od_dataInNotify_nextPeriod`	Function pointer. See separate callback contract.
`od_dataInNotify_free`	Function pointer. See separate callback contract.
Returns	
Preconditions	Online detection sub-system is stopped.
Notes	

Interface	`od_dataInNotify`
Method name	`od_dataInNotify_getNumRecords`
Purpose	Callback function to allow online detection sub-system to ascertain the number of records within the current period.
Parameters	
Returns	Number of records within period.
Preconditions	
Notes	

Interface	`od_dataInNotify`
Method name	`od_dataInNotify_getCallRecord`
Purpose	Callback function to allow the online detection sub-system to acquire a handle to the specified call record within the current period.
Parameters	
`record`	Index of record.
Returns	Handle to call record.
Preconditions	
Notes	See also `od_dataIn_free`.

Interface	`od_dataInNotify`

Method name	`od_dataInNotify_nextPeriod`
Purpose	Callback function to signify that the online detection sub-system has finished processing the current period.
Parameters	
Returns	
Preconditions	
Notes	This function must block until the next period is available and the data stream is ready to provide access to records for the next period.

Interface	`od_dataInNotify`
Method name	`od_dataInNotify_free`
Purpose	Callback function to allow the online detection sub-system to inform the data stream that it has finished with a handle to a call record. Facilitates memory management; may be stubbed if not required.
Parameters	
`callRecord`	Handle to record to be freed.
Returns	
Preconditions	Record has been previously acquired (see `od_dataIn_getCallRecord`).
Notes	

Interface	`od_detectorIn`
Method name	`od_detectorIn_loadDetector`
Purpose	To load a trained detector into the online detection sub-system.
Parameters	
`pDetector`	Handle to memory location containing packed detector to be loaded.
`size`	Number of bytes to read.
`uid`	Externally allocated unique identifier for this detector.
`detectionType`	Determines whether an anomaly indication will be raised on a match (if zero) or a mismatch (if non zero).
`threshold`	Determines the level of match/mismatch at which an anomaly indication will be raised. Valid values are greater than 0 (interpretation to be defined at design time). A negative value causes this argument to be ignored and the default value (to be specified at design time) to be used.
Returns	Return code indicating success/failure.
Preconditions	Trained detector available for loading. Detector with matching identifier is not loaded.
Notes	There is no guarantee how quickly the sub-system will respond. This function will block until the detector is loaded.

Interface	`od_detectorIn`
Method name	`od_detectorIn_unloadDetector`
Purpose	To unload a detector from the online detection sub-system.
Parameters	
`uid`	Identifier of detector to unload.
Returns	Indicator of success/failure.
Preconditions	Detector with matching identifier currently loaded.
Notes	There is no guarantee how quickly the sub-system will respond. This function will block until the detector is unloaded.

Interface	`od_anomaly`
Method name	`od_anomaly_setAnomalyNotify`
Purpose	To set a callback function to be used by the online detection sub-system for notification of anomalies in the online data stream.
Parameters	
`anomalyNotify_anomaly`	Function pointer. See separate callback contract.

Returns	
Preconditions	Online detection sub-system is stopped.
Notes	

Interface	`od_anomalyNotify`
Method name	`od_anomalyNotify_anomaly`
Purpose	To notify external system of the detection of an anomaly in the online data stream.
Parameters	
`uid`	Identifier of detector causing anomaly indication.
`detectionType`	Type of detection. Zero indicates match, non-zero indicates mismatch.
`threshold`	Threshold set for this detector.
`actual`	Actual value of indication (less than threshold for match, greater than threshold for mismatch).
`subscriber`	Identifier of the first subscriber within the current set of call records.
Returns	
Preconditions	
Notes	

Interface	`od_detection`
Method name	`od_detection_run`
Purpose	To initiate the online detection process.
Parameters	
`od_detectionNotify_error`	Function pointer. See separate callback contract.
Returns	Indicator of success/failure.
Preconditions	Input source and anomaly output connected (configuration is optional). At least one detector loaded. Online detection sub-system is stopped.
Notes	Callback function `od_detectionNotify_error` may be set to `null`, in which case it is ignored. Otherwise, this function will be called on each occurrence of an error in the detection process.

Interface	`od_detectionNotify`
Method name	`od_detectionNotify_error`
Purpose	Hook to acquire error notifications from detection process.
Parameters	
`error`	Error indicator (same format as return codes).
Returns	
Preconditions	
Notes	

Interface	`od_detection`
Method name	`od_detection_stop`
Purpose	To interrupt the detection process.
Parameters	
Returns	
Preconditions	Online detection sub-system started.
Notes	If detection process has been initiated, all progress will be lost, and the sub-system restored to a state ready for detection to begin. There is no guarantee how quickly the sub-system will respond – this call will block until the sub-system is ready.

1.9 Quality of Service requirements

This section considers the quality attributes of the FD/BP AI module. It should be noted that the purpose of this development is largely for proof-of-concept, to explore whether the selected AI techniques can add value to the network management domain. Hence, quality of service is not a primary concern. Nonetheless, quality attributes will be evaluated within the project.

1.9.1 Constraints and limitations

AI is not an exact science. Therefore, the results are, to some extent, unpredictable, and the system may raise false alarms for some conditions, and miss behaviours of interest for others. The effectiveness of AI techniques depends not only on the nature of the techniques, but also on the quality and quantity of the data, particularly that used for training. Further, learned structures are difficult for a user to understand and are not generally able to provide a chain of reasoning. Hence, the benefits of automated learning and generalisation come at the cost of non-determinism and lack of interpretability. The purpose of this development is to examine the ability of one or more selected AI techniques to meet some of the challenges of FD and BP, hence, the exploratory nature of this development should be considered during evaluation. The primary constraint comes from the lack of domain knowledge, specifically, the limited information available from the operator concerning the nature of fraud classes. The operator assumes that many more fraud classes exist within their environment than the single example included within the sample data, and, whilst evidence from the literature would support this view, the operator is not able to provide details. Hence, robust evaluation is challenging. However, an approach for extracting domain knowledge from the unlabelled data has been developed. This is a time consuming process that requires domain expert involvement, but allows the deployed techniques to move up the scale from unsupervised data mining, through anomaly detection to specialised misuse detection.

1.9.2 Scalability

In terms of proof-of-concept, scalability is not a primary concern, however, it is important that the selected techniques c*ould* be scaled to facilitate near real time analysis of full traffic on a complete network. Naturally, at this stage it is difficult to predict the run time of the finished AI module based only on the prototype system. However, since training is an offline process, the concern here is the execution time of the feature extraction and detection – data delivery is the concern of the integration framework and mediation layer. Consider the scenario of a hypothetical network with 15 million subscribers, all active during a given week. In order to process CDRs at the subscriber level, the detection sub-system must be able to perform feature extraction and run the loaded detectors at a rate of 25 subscribers per second. Performance measurements on the research prototype suggest that the execution time of individual detectors is very short, and the bottleneck may then be data delivery (outside of the scope of concerns of the AI module) and feature extraction; however, it does depend very much on the number of detectors loaded. Should performance increases be required, multiple instances of the AI module could be deployed, to exploit SMP and/or multicore hardware platforms. The selected approach is naturally data parallel, and can be scaled across clusters or other distributed platforms with ease, provided that the mediation layer / integration framework is capable of spawning independent instances of the module and distributing call records according to, for example, subscriber identifiers. The module will be implemented in C/C++ for performance reasons, and profiling tools may be used to target optimisation efforts should that be necessary.

One way to manage the mass of data, is to identify groups of subscribers that may be processed by the same detectors. The data mining process, described in Section 1.6.4, may be used to select groups of subscribers based on the similarity of their calling behaviours. These may then be processed without configuration / detector changes, thus improving overall throughput. Further, as domain knowledge is built up, the system may move from a largely anomaly detection basis, to a more misuse detection basis. Such a shift means that the same detectors may be applied to more (or possibly all) subscribers, further reducing the need for configuration / detector changes.

1.9.3 Reliability

Reliability concerns will be dealt with through rigorous software development processes and thorough testing, including stress testing. Stress testing can be facilitated through the use of large files of prepared CDRs, or by the development of a simple CDR generator, creating large quantities of randomised (synthesised) CDRs for analysis. Leak tracing tools will be deployed, if needed, to identify potential memory management issues. The required MTTF (defined in the requirements document, D2.4.1) is 7 days, with a maximum downtime of 30 minutes per week. This is not a particularly onerous requirement, provided that there are no memory management issues, either within the module, or the associated parts of the MDS operating on the same platform, and that thorough stress testing is performed to eliminate short MTTF defects. The latter point is important, because research has shown that many testing strategies are not sensitive to MTTF, and much effort is therefore spent removing defects that would not violate the quality of service constraints. Indeed, MTTFs tend to be distributed exponentially across several orders of magnitude such that the percentage of total defects within the software having an MTTF of less than 7 days may be considered to be particularly small (*e.g.* <<1%).

Since the module does not record state information from the previous processing cycle for use in the next, following a failure, it can be brought back online as soon as the system is available (*e.g.* after a reboot of the server, if necessary). The integration framework should ensure that the data feed can be rolled back to the previous period, to avoid missing an analysis step. Further, as suggested in the requirements document, multiple instances may be deployed on backup servers.

1.9.4 Effectiveness

According to the requirements document, D2.4.1, the system should aim to perform within 50% false positives and 10% false negatives, which corresponds to a detection rate of 83% and a false alarm rate of 36%. However, it was acknowledged that, in an exploratory system such as this, it is difficult to estimate whether this could be achieved in the proof-of-concept version. A requirement to support retraining was expressed, to enable an attempt to reduce the number of false positive and false negative indications. The detection rate is certainly challenging, especially given the lack of domain knowledge concerning misuse classes and their properties. The false alarm rate is less of a concern, although raising the alarm threshold to minimise false alarms may have some impact on the detection rate. In any event, it is evident that the operator's current understanding of fraud characteristics will not initially support such detection/false alarm rates, and the system must rely on the process of building domain knowledge, over time, following, for example, the process described in Section 1.4.1. Ultimately, with this hybrid approach, it is envisaged that such rates may indeed be achieved, particularly when misuse detection techniques become feasible. The caveat, of course, is that the behavioural properties present in the extracted feature vector will permit the separation of classes (*i.e.* that the irreducible classification error is significantly within the performance thresholds). These properties will need to be assessed during the evaluation of the MDS.

The processes surrounding expert intervention within the training and detection phases are particularly important here. Adopting the anomaly detection approach results in a system prone to false positives due to an incomplete picture of normality. This can be improved, over time, by incorporating the records from such false alarms into the training set for the anomaly detector, thereby attempting to correct such errors for the future. Adopting the misuse detection approach results in a system prone to false negatives, due to the inability to detect previously unseen misuse classes, where they differ markedly from the learnt class(es). Hence, the combination of both approaches is needed, with appropriate intervention from domain experts, to refine the knowledge repository in respect of training data for anomaly detection and known misuse classes for misuse detection.

Data mining will be a useful tool in improving the detection rates and false alarm rates, since it allows the tailoring of detector training to subscriber groups with particular behavioural characteristics. This results in more homogenous training data, and hence potential improvements in the discrimination between classes. Again, the utility of this approach will need to be assessed during the evaluation of the MDS.

The setting of an appropriate threshold for detection is not difficult, provided that the training data is representative. It has been suggested that this threshold be based on, for example, the peak output on the training data (assuming that data to be entirely normal), with a suitable margin added to minimise false positives. However, as domain knowledge is built up over time, it will be possible to be more 'intelligent' in the setting of this threshold.

For example, with samples of known normal and abnormal data, it would be possible to optimise the trade-off between detection and false alarms. With data mining to identify subscriber groups with similar behaviours, this may result in further improved separation of classes.

1.9.5 Methods of evaluation

The evaluation of the effectiveness of the FD/BP AI module is challenging as it considers two inseparable properties:

a. The ability of the AI solution to deliver the target detection and false alarm rates.

b. The nature of the data, that determines whether the information content will allow the AI solution to meet the performance targets.

Hence, a failure to meet a given target may or may not reflect a deficiency within the AI module. For example, if a given fraud class cannot be distinguished from otherwise normal behaviour based on the features available, this is a problem with the data, rather than the AI module. It is very difficult to separate these concerns. Further, meeting the target detection and false alarm rates requires that the class separation (*i.e.* the difference in properties between normal and abnormal behaviour) exceeds the natural variation in normal patterns of subscriber behaviour. That is, even if the system is capable of distinguishing between normal and abnormal behaviours, the noise within the normal profiles may result in false positive indications or, if the threshold is increased to reduce or eliminate them, false negative indications. Thus, the recommendation would be to use the output of the AI module as a 'priority list' for investigation.

Given the lack of domain knowledge, it is challenging to establish an effective evaluation scheme. The operator, for example, is not able to provide misuse cases for anything other than FCT fraud. This prevents empirical studies that might otherwise gain a measure of the effectiveness of the AI module, or even some confidence in the output. Three strategies are proposed, therefore, for evaluation:

a. During the evaluation phase, the operator will deliver a large number of subscriber-weeks worth of data to the module. The prioritised outputs of the AI module will be sampled to assess the reliability of the alarms raised. Misclassified outputs will inform the retraining of the AI, and then the AI will be reassessed. Several such cycles may be employed to improve the performance.

b. Synthetic data sets, such as that used for the feasibility study in Section 1.4.3, will be generated to explore the sensitivity of the AI module to variations in various features. This will provide an indication of the discrimination possible. The operator will need to provide information on the sort of deviations that might be indicative of fraudulent usage, based on their domain knowledge. This will inform the synthesis of the test data.

c. A cost / benefit analysis will be performed to assess the business potential of the AI module. This will involve projections of alarm rates from the sample data to a complete network, and an estimate of the time and cost involved in expert verification of results, and the savings possible from improved revenue protection. The operator will need to provide the information required to produce such a model. This form of evaluation is arguably far more important than the actual detection and false alarm rates, as it will indicate the value added to the operator's business by the use of the AI module, and the implications of doing so.

The data mining process is also difficult to evaluate since there is no intrinsic notion of success or failure in data mining. Since the purpose of data mining is to identify naturally occurring patterns in data, it is entirely possible that there may not be any readily discernable patterns based on the features available, or that patterns that may be discerned may have no useful interpretation in the domain. The proposed evaluation process is therefore based on the notion of improving the homogeneity of training data. One or more detectors will be trained on a given data set. The mining, with pre-trained detectors, will be performed to identify subscriber groups. The corresponding data set will be filtered according to subscriber group and new detectors trained. The resulting detectors should show improved training accuracy and discrimination ability. These will be used to mine the data, again, and assess the ability of this process to distinguish differing behavioural profiles. Such efforts may well also feed into the evaluation of the AI module in terms of detection and false alarm rates. The data mining results will also be inspected by domain experts, in an attempt to assess the utility from a business process perspective.