Software Systems Modelling

Dr Keith Phalp (and others...)

Alternative Designs

- Just as an example of alternative design approaches this example, from a state of the art EC funded international project uses a combination of approaches (all of which are non-UML).
- The requirements document and the design document both use the same basic approach.
- The design expands upon the requirements.
- Modelling notations include Role Activity Diagrams and DFDs, linked by shared functions.
- An API is also produced.

Background Approach

1) Behavioural process models (depicted as Role Activity Diagrams, RADs), show the actions that are required from the domain from the major roles (either people or sub-systems), the interactions among those roles, and the dependencies and interdependencies of those actions. The textual narrative to accompany the Role Activity Diagrams cross references those activities (actions and interactions) with the requirements list in Section . Requirements are denoted DTx and ODx, for DT and OD sub-systems, respectively, where x is the specific numbered requirement.

2) A procedural model (depicted as a Yourdon data flow diagram, DFD) shows the system boundaries for FD and 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 and BP, and the data that must cross sub-system boundaries, and which must, therefore, be specified.

System View and Sub-systems

- Fraud detection and business process aspects of the MDS are both centred on the processing of Call Data Records (CDRs), and the identification of patterns that represent some variation in behaviour from accepted norms.
- These patterns must be identified for later interpretation by experts that are able, on the basis of domain knowledge and/or information gathered from other sources (such as performance and fault data), to ascertain whether the patterns are: indicative of fraud, indicative of a change in user behaviour that is of importance to the operator's business processes OR incidental, and of no interest to the operator
- 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 and BP sub-systems of the MDS.
- Since the MDS focuses on the application of Artificial Intelligence (AI) techniques to various aspects of the operator's business, there is a clear need for two sub-systems: one for training detectors (that is, machines for the detection of variations in behaviours), and one for online detection (that is, the deployment of trained detectors). In addition, a data mining process is also considered, as a useful way of identifying appropriate training data, as well as supporting directly the operator's business processes (by visualising/analysing patterns of calling behaviour).

Detector Training

- The purpose of detector training (DT) is to create a computational machine (referred to as a detector) capable of recognising a usage profile based on example.
- Training is based on building a profile of behaviour over a number of defined intervals (in the prototype system this is to be restricted to weekly intervals), where each interval comprises zero or more individual call records for one or more users, selected by a domain expert.
- An AI engine is 'learns' the essence of user behaviour during these periods, and develops a detector that is capable of offering an indication of variation...
- 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. Patterns of behaviour may be indicative of fraud or of behaviours of interest to the operator's business processes.
- 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.

Detector Training

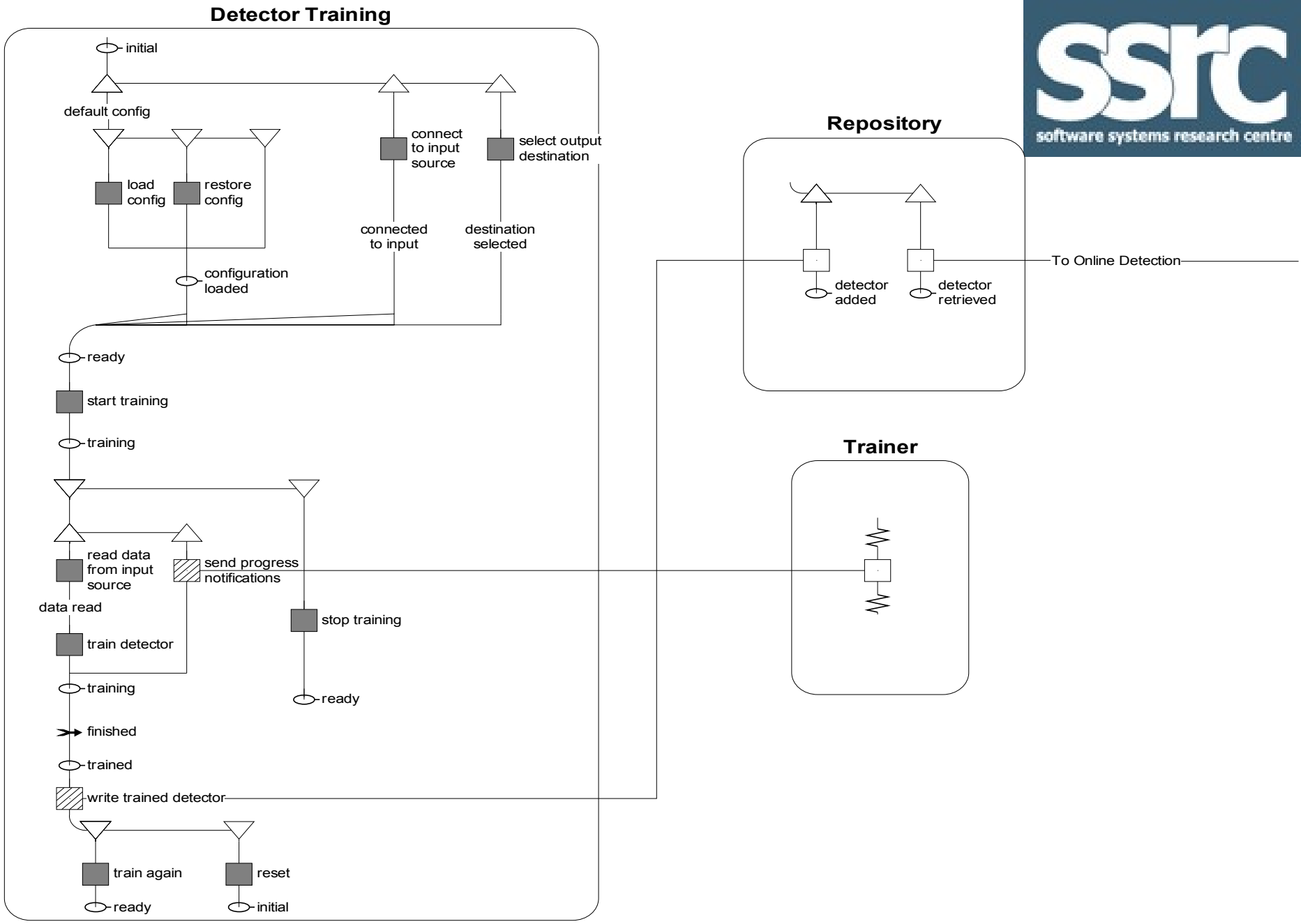
- On commencement (*initial*), there are three parallel threads.
  - 1) Configuration, within which there are three choices. These are either *load config* (DT1) or *restore config* (DT2), or no action, which thus implies that the current configuration will be kept (on commencement this is the default configuration). Either of these threads results in a configuration being loaded (configuration loaded state).
  - 2) *Connect to input source*, for training data (DT3).
  - 3) *Select output destination*, for trained detectors (DT4).
- Once all three parallel threads are complete, that is, *configuration loaded*, *connected to input* and *destination selected*, a *ready* state is reached and the *start training* (DT5) action can be carried out, resulting in the role being in a *training* state. From here there are two main choices. One is the continuation of training (until *finished*); the other is to *stop training* (DT6). Continuation includes two parallel threads. The first thread is to *read data from input source* (DT7). Once data is read (*data read*), training the detector on the input data (*train detector*, DT8), can commence. The *training* state is then reached again, and the sequence of reading and training continues until training is *finished*, when the *trained* state is reached. From this state the *write trained detector* interaction writes the detector (DT10) to the output destination (modelled as the *Repository*). Having written the detector the role is then *ready* to being the training cycle again. The second thread allows for progress reporting (DT9), shown as an interaction with the *Trainer*. Finally, there is the choice to *train again*, returning to the *ready* state, or to *reset*, which returns to the *initial* state, so that configurations and connections can be chosen again.

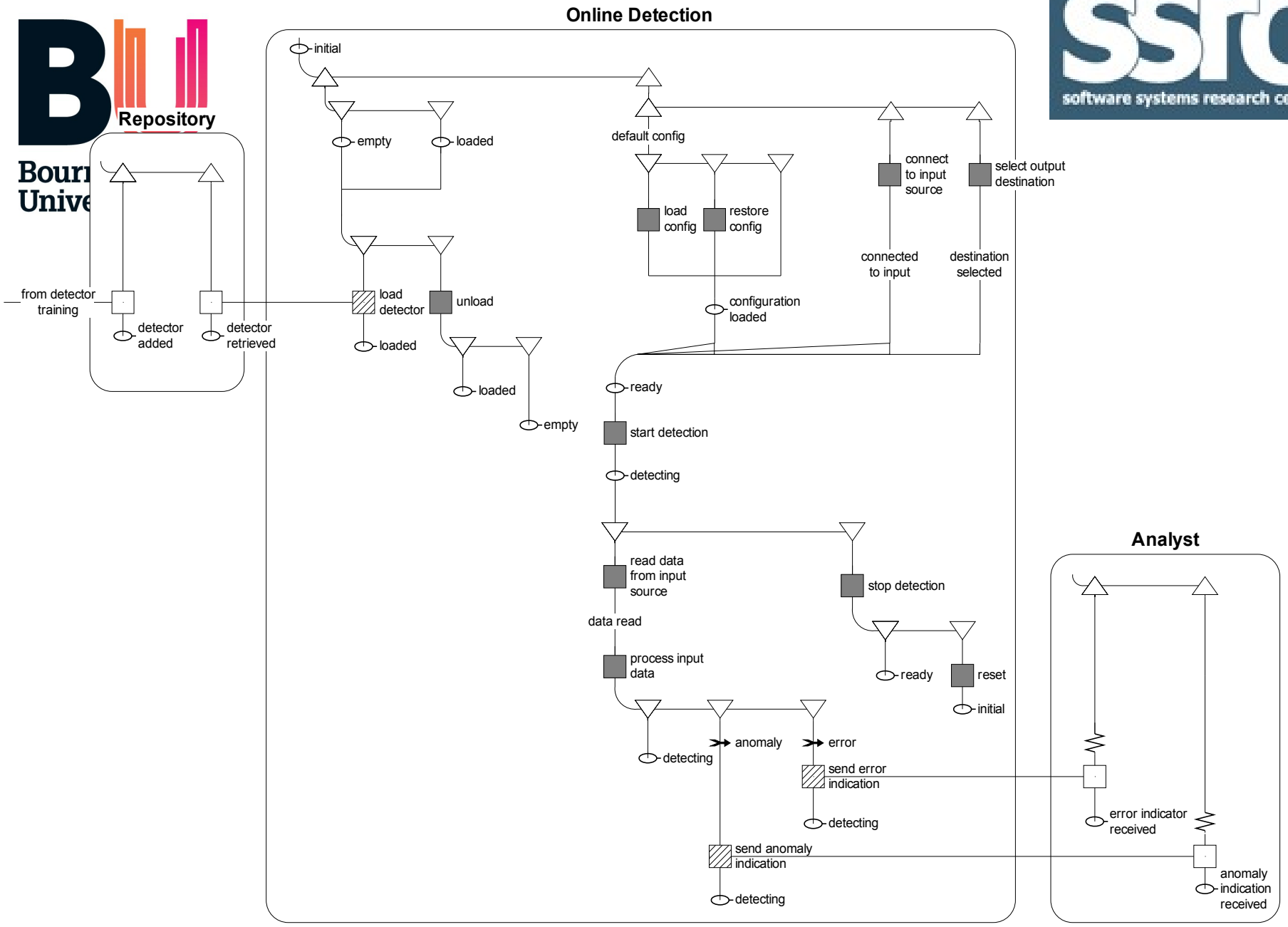
On-line Detection

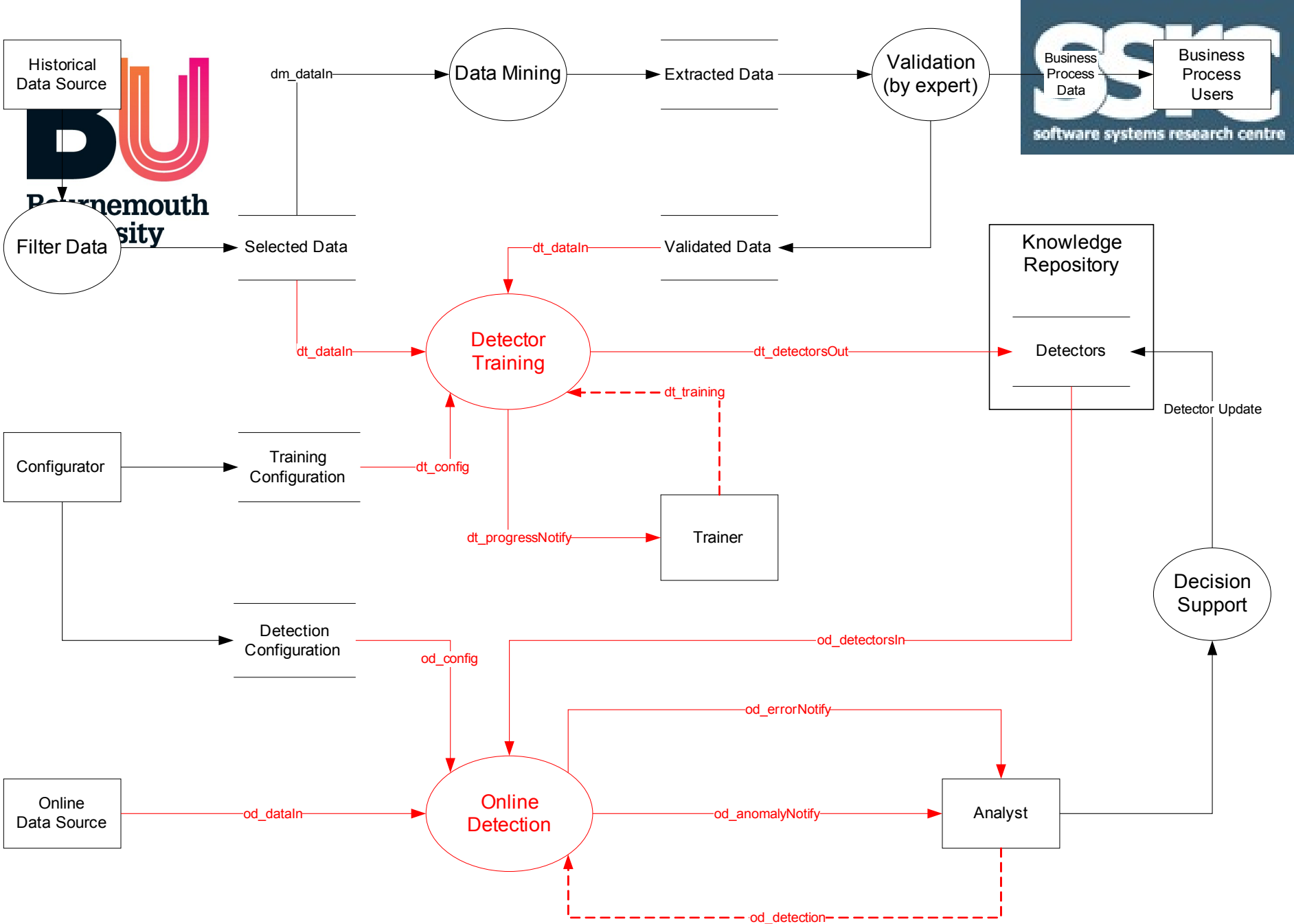
- Online detection (OD) is concerned with the deployment of one or more previously trained detectors.
- These detectors should be capable of raising anomaly indications on the occurrence of either a match or mismatch between the learnt behaviours and those present in the online data. That is, a detector may be trained to recognise fraudulent behaviour, or normal behaviour.
- The detector's purpose is to extract the essence of the behaviours and provide an indication of the variation between the training set and the online data.
- The OD sub-system should, therefore, offer the option of processing this for a match or mismatch, in effect implementing misuse detection and anomaly detection, in the accepted parlance of the field.
- Multiple detectors may be applied simultaneously, to characterise incoming data in different ways, such as for different classes of fraud.

On-line Detection

- The Online Detection role is broken into two higher level parallel structures, which involve: The loading and unloading of detectors; the configuration and connection, then subsequent detection. Note that loading and detection are separate, and, hence, detection may commence irrespective of detectors loaded. However, in practice one may wish to restrict the detection process such that its commencement and continuation is conditional on one or more detector(s) being loaded. First, the detectors thread is considered. Either some detectors are already loaded, or the detectors are empty. In either case, the loading of a (trained) detector (OD3), takes the role to the *loaded* state. Note that this involves an interaction with the *Repository*, from where trained detectors are retrieved. Alternatively, a detector may be unloaded (OD4). Depending on the number of detectors loaded, the *loaded* or *empty* state would be reached, and so on (note that an unload attempt on the *empty* state will simply return to the *empty* state). As with the DT model, the setup thread, which deals with the prerequisite for the OD to be ready for detection, involves three parallel threads: Configuration, within which there are three choices. These are either *load config* (OD1) or *restore config* (OD2), or no action, which thus implies that the current configuration will be kept (on commencement this is the default configuration). Either of these threads results in a configuration being loaded (configuration loaded state). *Connect to input source* for online data (OD5). *Select output destination*, for anomaly indications (OD6).
- It is the completion of the setup thread that moves *Online Detection* to the *ready* state. From here *start detection* (OD7), causes a move to the *detecting* state. Within *detecting* there are two main (choice) threads. The first consists of the reading (OD9) input data and processing using loaded detectors (OD10). If nothing is found within the data, then detecting simply continues, with the next read and process cycle. If an anomaly is found, (event *anomaly*) then an anomaly indication is sent to the *Analyst* (OD11), and then detecting continues. If an error is found (event *error*), then an error indication is sent to the *Analyst* (OD12), and again detecting continues. The second thread allows for detection to be stopped (OD8), at which point the sub-system is either able to return to the *ready*, to resume detection, or to be reset returning to its *initial* state), *e.g.* so that the setup can be changed.


Detector Training
initial
default config
load config
restore config
connect to input source
select output destination
connected to input
destination selected
configuration loaded
ready
start training
training
read data from input source
send progress notifications
data read
train detector
stop training
training
ready
finished
trained
write trained detector
train again
reset
ready
initial
Repository
detector added
detector retrieved
To Online Detection
Trainer


Online Detection
Repository
initial
empty
loaded
default config
load config
restore config
connect to input source
select output destination
connected to input
destination selected
configuration loaded
from detector training
detector added
detector retrieved
load detector
unload
loaded
loaded
empty
ready
start detection
detecting
read data from input source
stop detection
data read
process input data
ready
reset
initial
detecting
anomaly
error
send error indication
detecting
send anomaly indication
detecting
Analyst
error indicator received
anomaly indication received


Historical Data Source
dm_dataIn
Data Mining
Extracted Data
Validation (by expert)
Business Process Data
Business Process Users
Filter Data
Selected Data
dt_dataIn
Validated Data
Knowledge Repository
Detectors
Detector Training
dt_dataIn
dt_detectorsOut
dt_training
Detector Update
Configurator
Training Configuration
dt_config
dt_progressNotify
Trainer
Decision Support
Detection Configuration
od_detectorsIn
od_config
od_errorNotify
Online Data Source
od_dataIn
Online Detection
od_anomalyNotify
Analyst
od_detection

Training Requirements

The trainer must be able to:

DT1. Load a configuration file
DT2. Restore default configuration
DT3. Connect to an input source
DT4. Select output destination for trained detectors
DT5. Start training cycle
DT6. Stop training cycle

While the training cycle is running, the detector training sub-system must:

DT7. Read data from input source, as required for training process
DT8. Train detector on input data
DT9. Send progress notifications to trainer
DT10. Write trained detector to output destination

Detection Requirements

The analyst must be able to:

OD1. Load a configuration file
OD2. Restore default configuration
OD3. Load a (trained) detector
OD4. Unload a detector
OD5. Connect to an input source
OD6. Select output destination for anomaly indications
OD7. Start detection process
OD8. Stop detection process

While the detection process is running, the online detection sub-system must:

OD9. Read data from input source
OD10. Process input data with loaded detectors
OD11. Send anomaly indications to analyst
OD12. Provide analyst with indications of errors encountered during processing

Bournemouth University

Summary

ssrc software systems research centre

- Brief overview of how different models can be used depending on the context.
- This is the requirements phase models.
- The design document has more detailed RADs and DFDs, along with UML component models, functions described, and a complete description of the systems design.
- Of course, this is not an argument for using RADs, rather an illustration of choice. That is, choose the notation(s) for the job at hand.
- In essence, this 'horses for courses' argument suggests something like a 'problem frames' approach to Software Systems Modelling.