What's new in NetRexx 5.10

Marc Remes

remesm@gmail.com

Agenda

- Native concurrency with the new GO instruction
- Concurrency synchronisation and communication with RexxChannels
- Default methods on interface classes
- Language Server Protocol support

The GO instruction

- Simplifies multi-threading and concurrency in NetRexx
- `GO expression;`
  - where expression is a method call
- The method is started in a new thread and can have any number of arguments
- `return;`
  - ends the thread, any return value is discarded
- Exceptions during method thread execution are printed on stdout

GO easy

```
class goeasy

  method main(argwords=String[]) static
    loop i = 1 to 5
      go easy(i)
    end

  method easy(a) static
    say 'hello 'a' from 'Thread.currentThread().getName()
```

GO implementation

- The GO instruction gets wrapped as a Java Thread class
- See src/org/netrexx/process/NrGo.nrx
- Thread scheduling is nondeterministic, controlled by the OS/JVM run-time
- Control control concurrency, synchronization and inter-thread communication with RexxChannels

RexxChannels

- A communications mechanism between NetRexx threads
- New NetRexx type, see src/netrexx/lang/RexxChannel.nrx
  - `write(Object)`
  - `read() returns Object`
  - `close()`
- Unbuffered
  - write blocks until a corresponding read occurs
  - read blocks until something is written.
- Buffered
  - write operations complete up to the given capacity. When a channel is full, write blocks until a reader consumes an item.
  - When a channel is empty, a read operation blocks until something is available.

RexxChannel specifics

- Differences between buffered and unbuffered channels

Feature	**Unbuffered channel**	**Buffered channel**
Size	No size specified	Explicit capacity
Write	Blocks when no reader waiting	Blocks when channel full
Read	Blocks when nothing written	Blocks when channel empty
Close	can read until empty, then IOException on further reads/writes	
Best for	Synchronization	Communication

RexxChannel Use cases

- Synchronous communication with unbuffered channel
  - `examples/go/pingpong.nrx`
- Resource management and structured parallelism with buffered channels
  - `examples/go/workerpool.nrx`
- Pipelines with buffered channels
  - `examples/go/pipeline.nrx`
- Resource limiting with buffered channels
  - `examples/go/resourcelimit.nrx`

Default methods

- Further implementation of Java 8 default methods
  - methods with actual implementation on abstract interface classes
    - Called when implementing class does not provide the method
    - Allows to add a method on interface class without the need to update all implementing classes
- Was previously implemented but not documented
- Parser and image loader updated to recognise `default` modifier, but only on `interface` classes

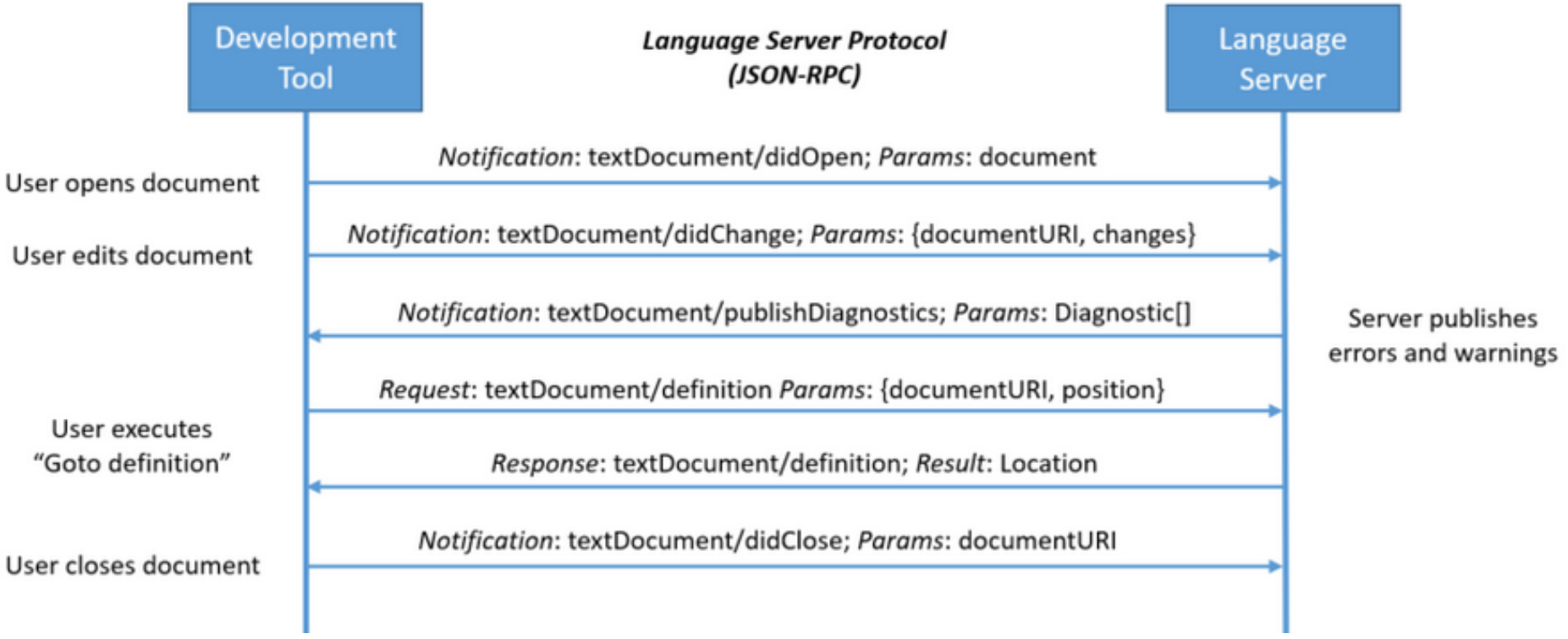
Default methods

- Improved error handling
  - Two interface classes with each identical default methods, not implemented by implementer
    - `Multiple default interface methods, class 'c' must implement method m`
  - Two interface classes with identical methods but one default, not implemented by the implementer
    - `Class is not abstract, yet it does not implement method 'm' from interface class 'i'`

Language Server Protocol

- Microsoft defined protocol to support compile errors, auto-complete, go to definition, find all references etc.
- The language server which runs in a separate process, communicates with the IDE using the JSON-RPC protocol
- LSP standardises the message format and message flow
- Language agnostic – can be written in any language
- Can be reused by any IDE that supports LSP
- Inter-process communication by default is over stdin/stdout

Language Server Protocol



lsp4nrx

- LSP implementation for NetRexx, written in NetRexx
- Source code in src/org/netrexx/lsp
- Packaged as org.netrexx.lsp
- Started as `java org.netrexx.lsp.NetRexxLspServer`
- Optional arguments:
  - `--onChange=nnnn` delay in ms before interactive parsing (or `off`)
  - `--log` emits JSON-RPC messages on stderr
  - `--stdio` default communication
  - `--sock` to pre-start language server while debugging

lsp4nrx supported requests

- IDE to LSP Server
- Requires response
- Implemented in lsp4nrx
  - initialize
    - Response with capabilities textDocumentSync and documentSymbolProvider
  - initialized
    - After reception of initialize response
  - shutdown
    - Response with null JSONObject
  - textDocument/documentSymbol
    - Response with hierarchy of class and methods for 'outline' or 'structure' by fast pass1 compilation

lsp4nrx supported notifications

- IDE to LSP server
  - exit
    - IDE asks server to exit
  - textDocument/didOpen
    - server stores source file in cache and schedules diagnostics
  - textDocument/didClose
    - server removes source file from cache and clears diagnostics
  - textDocument/didChange
    - Server updates cache and schedules diagnostics
  - textDocument/didSave
    - Server updates cache and schedules diagnostics

lsp4nrx supported notifications

- LSP server to IDE
  - textDocument/publishDiagnostics
    - compile NetRexx source from file (didOpen/didSave) or from memory (didChange)
    - send JSON-RPC with line and column information of warnings/errors

lsp4nrx plug-ins

- Each IDE has its own particular way to incorporate an LSP server
- Plugin packages created for
  - Microsoft's vscode
    - Typescript
  - JetBrains' IntelliJ
    - Java
  - eclipse by the Eclipse Foundation
    - Java

lsp4nrx plug-in options

- All plug-ins have the same options available
  - onChange, number of ms to wait to start diagnostics
  - log, log full JSON-RPC messages on stderr
  - classpath, list of directories or JAR files containing dependent classes

Next

- Further development of the LSP implementation
  - Hover
  - Go to Definition
  - Code completion
  - Debug