Communication Systems in ClawZ

Michael Vernon[1], Frank Zeyda[2], and Ana Cavalcanti[2]

[1] QinetiQ, Cody Technology Park, Farnborough, Hampshire, GU14 0LX, U.K.
mrvernon1@qinetiq.com
[2] University of York, York, YO10 5DD, U.K.
{zeyda,ana}@cs.york.ac.uk

Abstract. We investigate the use of ClawZ, a suite of tools for the verification of implementations of control laws, to construct formal models for control systems in the area of communications and signal-processing intensive applications. Whereas ClawZ has been successfully applied to verify control components in avionic systems, special requirements need to be identified and addressed to extend its use to the aforementioned application domain. This gives rise to several extensions, which we explain and subsequently validate by constructing the Z model of a software-defined radio communication device. The experience reported provides insight into general issues surrounding the use and extension of ClawZ.

Key words: control laws; signal processing; formal models; Z; Simulink

1 Introduction

Control law diagrams are a graphical notation widely used by engineers to specify the behaviour of control systems. In industry, the commercial tool Simulink by MathWorks [11] is a de-facto standard for the design of control diagrams. Roughly speaking, control diagrams consist of blocks that carry out elementary functions, and wires that transmit data values between those blocks. Diagrams communicate with the environment through designated input and output port blocks. They may also exhibit structure in which the functionality of basic blocks may itself be described by virtue of lower-level diagrams. Additionally, Simulink provides a comprehensive library of blocks and supplementary toolboxes to support the specification of control systems for particular application domains.

Here, a formal approach to verification of implementations of diagrams is advocated. If we cannot rely on automatic code generators to ensure correctness, for instance, because code has to be optimised, the ClawZ suite of tools [4,2] can be used to construct a proof of the correctness of an Ada implementation. ClawZ is a highly automated set of utilities for use in industrial-scale projects.

Software-defined radios have recently gained popularity [10]; they perform most of their signal-processing operations in software, for instance, on a personal computer or DSP. This allows them to support simultaneously many communication standards, each requiring specific demodulation and decoding techniques, in

a single integrated piece of hardware [8]. Their potential use in the military sector and other safety-critical areas highlights the need for formal verification [6].

Although ClawZ provides flexible mechanisms to configure and extend it for use with a wider class of diagrams, it lacks support for control laws that are typically found in the design of signal-processing and radio communication devices. These diagrams differ from others in that they require support for complex number arithmetics. Simulink is oblivious to this distinction due to the effective polymorphism of block behaviours, but the formal model needs to reflect the difference. Similarly, ClawZ does not support matrices as signal types.

A second problem is that many blocks commonly found in signal-processing models, such as filters, modulators, Fourier transformers, and so on, are not part of the library of translatable blocks in ClawZ. Extending this library requires Z models to be developed for the blocks according to their function.

Here we report on work that realises the above enhancements, validates them in the context of a software-defined radio case study, and thereby provides evidence that it is possible to use ClawZ for generating formal models for this class of diagrams. This widens the applicability of ClawZ, and sheds light into a few issues of the ClawZ approach to building Z models, which imposes limitations on automation, namely due to insensitivity to signal types.

In Section 2 we provide further details on ClawZ and identify requirements for using it for signal-processing control systems. Section 3 discusses our extensions of ClawZ; Section 4 validates them using our case study. Finally in Section 5 we draw our conclusions and suggest future work.

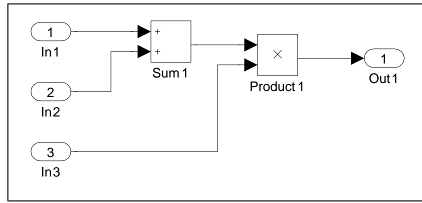
2 Preliminaries

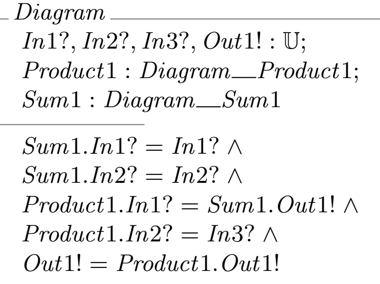
After providing an overview of ClawZ in Section 2.1, we discuss the main features of communication control systems in Section 2.2.

2.1 ClawZ

The ClawZ verification process involves the construction of a Z model for a Simulink diagram acting as a specification of behaviour to which an implementing Ada code has to adhere. ProofPower-Z [1,3], a theorem prover for Z based on higher-order logic (HOL), is used for mechanical proof. Correctness is established by an embedding of the refinement calculus into ProofPower-Z. It has been successfully used in industry, with a measured cost reduction of 20% in the certification of avionics systems. The fact that it can be adapted to other areas, and that the same high level of automation can be achieved by programming of proof tactics, makes this approach very attractive.

The ClawZ tools are bespoke and automated, tailored for engineers without in-depth knowledge of formal specification and proof. The tool that carries out the translation of diagrams into Z specifications is the Z Producer. To illustrate its approach, we consider the diagram consisting of a Sum and Product block in Fig. 1. The main schema in the specification generated for this diagram is also in





Diagram

$In1?, In2?, In3?, Out1! : \mathbb{U};$
$Product1 : Diagram__Product1;$
$Sum1 : Diagram__Sum1$

$Sum1.In1? = In1? \land$
$Sum1.In2? = In2? \land$
$Product1.In1? = Sum1.Out1! \land$
$Product1.In2? = In3? \land$
$Out1! = Product1.Out1!$

Fig. 1. Simple Simulink diagram consisting of two blocks.

Fig. 1. It introduces components for the input and output ports of the diagram, namely $In1?$, $In2?$, $In3?$ and $Out1!$; port variables are obtained by suffixing *In* and *Out* with the port number. The schema also includes components that characterise the behaviour of the blocks.

Whereas the $Product1$ and $Sum1$ components represent particular instances of the Product and Sum blocks, $Diagram__Product1$ and $Diagram__Sum1$ are schema types that encapsulate the behaviours of these blocks. Upon translation these types are introduced by associating them with suitable block definitions in the ClawZ block library. The library is contained in a separate ProofPower-Z theory acting as a carrier to hold those definitions. ProofPower theories are in essence collections of type definitions, constants, defining axioms and theorems.

The type $Diagram__Product1$, for example, is defined as $Product_M2$ where $Product_M2$ is the block schema specified in the library. Its definition is given below and follows the same conventions on port names.

Product_M2

$In1?, In2?, Out1! : \mathbb{R}$

$Out1! = In1? \ *_R \ In2?$

Above, the types of the ports are explicitly given, whereas in *Diagram* they are unspecified; $\mathbb{U}$ acts as a generic type to be inferred by the typechecker. The equations in the predicate of *Diagram* such as $Product1.In1? = Sum1.Out1!$ encode the wiring of blocks: each connecting wire results in one equality.

Block specifications may be functions yielding schemas too. This allows models to be parameterised by arguments set for the block inside Simulink. For example, the initial output of a unit delay block, which delays a signal by one cycle of the control system, is a parameter of its model in the library: a function from $\mathbb{U}$ to the schema type representing the block.

The Simulink diagrams to be translated by the Z Producer can be arbitrarily structured. For example, the diagram in Fig. 1 could itself appear as a block in a higher-level model. In this case *Diagram* would act in turn as a schema type for the aggregated component representing that block. ClawZ also includes additional schemas in the model that specify the behaviour of blocks for reset and hold cycles, that is when the signal value is either reset to some initial value, or

simply retained. Finally, an additional feature allows certain simple block models to be constructed on-the-fly rather than imported from the library.

The above only considers a simple example but in essence illustrates how the generation of Z models is carried out. A crucial aspect is that it is realised automatically, mostly as a syntactic transformation from the Simulink `MDL` file that gives the textual description of a diagram to the respective ProofPower-Z theory source file. This surfaces, for example, in that the types of port components in the *Diagram* schema are given as $\mathbb{U}$ due to no type information for signals being available that could otherwise be exploited to specify exact types.

2.2 Signal-Processing Features

The following techniques are particularly relevant for signal-processing models. Their support in ClawZ is the primary objective of our work. Communication devices heavily rely on them, but they are also relevant for the many applications that require digital image and sound processing or data compression.

Filtering Filtering in essence allows to shape signals by amplifying frequencies within desired ranges while suppressing others considered as noise or irrelevant. It is a fundamental operation within many signal-processing algorithms to extract information from signals and prepare them for further processing. The theory of digital filters is well-developed; basic classifications are Finite Impulse Response (FIR) filters and Infinite Impulse Response (IIR) filters. Filters are usually characterised by their dimension and filter coefficients. Such can either be specified statically, or, in adaptive filters, adjusted dynamically according to the minimisation of the error between a desired and actual signal.

Modulation Modulation is used, for instance, to transfer binary data over analog passband channels, and is geared towards the capabilities and characteristics of the channel to maximise the amount of information transmitted. Modulation is usually carried out before information is transmitted through a channel, and demodulation, its inverse, to retrieve the information upon reception.

Various approaches to modulation exist such as Quadrature Amplitude Modulation (QAM), Phase-Shift Keying (PSK), Frequency-Shift Keying (FSK), and others. Modulation gives rise to signals being interpreted in the complex number plane; an essential aspect for their support is hence complex arithmetics.

Encoders / Decoders The encoding of signals has the dual purpose of adding redundancy for error detection and correction, and encrypting signals that carry sensitive information, for example in military communication devices. It is usually carried out prior to modulation. An example we considered is the Trellis encoding of digital signals which is based on convolution codes. Those codes are frequently used in digital radio applications because of their favourable properties approaching theoretical limits for the amount of information that they transport through a lossy channel.

Fourier Transformation Fourier-transformation is a final operation of interest, playing an important part in analysing and compressing signals, for example in image or speech processing applications. Again, the Fourier-transform of

a real-valued signal is usually a complex function where phase and amplitude encode amplification and phase-shift of respective frequencies. Again this makes the support for complex numbers and their operators imperative.

3 Extension of ClawZ

In this section we explain in more detail how we extend ClawZ to support the signal-processing features that were outlined in the previous section. For this we first discuss the support for additional data types like complex numbers and matrices, and then report on additions made to the ClawZ block library.

3.1 Addition of Data Types

The subset of the Simulink notation that can be handled (modelled in Z) using ClawZ only admits scalars (of type real), and vectors (of type real). In communication-related control laws we often require to operate on scalars, vectors and matrices of complex numbers. They are used, for example, to encode the amplitude and phase of signals in the frequency domain as they occur in Fourier transforms or demodulation of signals. This suggested two fundamental extensions to the ClawZ tools: one is to deal with complex numbers as such, and another is to support complex vectors and matrices as data values being passed between the blocks of a Simulink diagram.

For integrating these extensions it makes sense to distinguish between two independent concerns: firstly to formalise them in the logic of the underlying theorem prover, ProofPower-Z, and secondly to extend the Z Producer to handle the new types in the translation of Simulink diagrams into formal Z models. There exists no comprehensive embedding of complex numbers in ProofPower-Z to our knowledge, however, a case study is available on the ProofPower web-pages that illustrates such an extension in principle; we pursue a similar approach being described in what follows. The complete set of definitions can be found in Vernon's MMath thesis [14], and the ProofPower-Z theory source is made available for download at `http://www.cs.york.ac.uk/circus/tp/tools.html`.

To formalise complex numbers we introduce a new axiomatic constant $\mathbb{C}$ as the set of tuples of real numbers: $\mathbb{C} \mathrel{\widehat=} \mathbb{R} \times \mathbb{R}$. This set acts as the *type* used for complex numbers. The first component represents the real part of the number, and the second, the imaginary part. ProofPower-Z supports real-number arithmetics by means of a collection of relations and functions that operate on elements of $\mathbb{R}$ such as $+_R$, $*_R$, $\leq_R$, and so on. The subscript highlights what type of value these functions expect, and a repository of axioms and derived theorems permits reasoning about formulae involving the operators. Similarly, we introduce a set of function definitions that operate on elements of $\mathbb{C}$.

Most of the axioms for operators directly correspond to familiar textbook definitions of those operators. For example, the following Z axiomatic definition

introduces multiplication of complex numbers.

$$\begin{array}{l} _ *_C _ : \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C} \\ \hline \forall z, w : \mathbb{C} \bullet z *_C w = (z.1 *_R w.1 -_R z.2 *_R w.2, z.1 *_R w.2 +_R z.2 *_R w.1) \end{array}$$

The dot operator selects the components of a tuple. For lack of space we will not further discuss the remaining operators but point to [14] for their definitions.

Vectors of complex numbers are characterised by sequences over $\mathbb{C}$ being similar to how ClawZ encodes vectors of real numbers in the Z model of diagrams. Again, a collection of useful operators is defined, for instance, to calculate the scalar product and the sum of vectors. Complex matrices, on the other hand, we characterise by sequences of sequences over elements from $\mathbb{C}$; each inner sequence represents one row of the matrix. It shall be noted that other characterisations are conceivable too, but at present we have no conclusive evidence to favour one over another. The following set introduces the matrix type formally in Z.

$$M_C \mathrel{\widehat{=}} \{m : \mathrm{seq}_1 (\mathrm{seq}_1 \mathbb{C}) \mid \forall i, j : 1..(\# m) \bullet \# m(i) = \# m(j)\}$$

A requirement here is that the inner sequences must all have the same length. We also exclude the limit case of zero-dimensional matrices by confining ourselves to non-empty sequences; in practice there is no need for such matrices.

As with vectors, there exists a number of specific operations applicable to matrices. The following definition, for instance, introduces multiplication of complex matrices. The number of columns of the first matrix $\# A(1)$ must be equal to the number of rows of the second matrix $\# B$ to apply the operator.

$$\begin{array}{l} _ *_{MC} _ : M_C \times M_C \nrightarrow M_C \\ \hline \mathrm{dom}(_ *_{MC} _) = \{A, B : M_C \mid \# A(1) = \# B\} \land \\ (\forall A, B : M_C \mid \# A(1) = \# B \bullet \\ \quad A *_{MC} B = \{m : 1..(\# A) \bullet m \mapsto \{n : 1..(\# B(1)) \bullet \\ \quad\quad n \mapsto \mathit{SumSeq}_C (\{k : 1..(\# A(1)) \bullet k \mapsto A(m)(k) *_C B(k)(n)\})\}\}) \end{array}$$

Multiplication is carried out according to the rule $C_{mn} = \sum_{k=1}^{K} A_{mk} * B_{kn}$ where m ranges over the rows of A, and n over the columns of B. The function SumSeq_C is introduced to calculate the sum of the elements of a complex sequence.

Other operators that have been formalised include matrix addition, transposition, complex-conjugate, and functions to extract the real and imaginary part of complex matrices. We also provided operators for scalar multiplication as well as multiplication of matrices with (complex) vectors of the correct size.

ClawZ Integration Whereas the formalisation of complex arithmetics and matrix algebra is not a difficult problem *per se*, more challenging proves to incorporate the new types and operators into the generation of Z models from diagrams. As previously mentioned the translation carried out by ClawZ is mostly a syntactic transformation process in that it does not utilise information about signal types; type information for signals is neither inferred nor exploited. (To

be accurate, there are situations where signal types are taken into account, that is, the synthesis of certain kinds of blocks, but this is not the general case.)

A problem in the translation arises due to the fact that Simulink blocks exhibit polymorphic behaviours, meaning the same type of block may process input signals of different types. For example, the Sum block can be used to add two scalar inputs, one scalar and one vector, two vectors, or even matrices if their dimensions agree. In the formal model such blocks must have different specifications, depending on their types of inputs. The problem is exacerbated in our work as we moreover have to consider operations on complex types.

To illustrate the above, consider several possible encodings of the Sum1 block in Fig. 1 as a Z schema. One may assume the inputs $In1?$ and $In2?$ to be real scalars, giving rise to $[In1?, In2?, Out1! : \mathbb{R} \mid Out1! = In1? +_R In2?]$ as its block schema. We may alternatively consider the inputs to be vectors, matrices, or complex scalars, giving rise to different characterisations. The problem does in fact surface even when translating 'conventional' diagrams, and the solution adopted by ClawZ is to support means of selectively determining what underlying formal model to use for particular blocks. Consequently, the user is sometimes required to analyse the diagram by hand when generating its Z model.

To address this problem in the context of our extensions, we propose an initial analysis of the Simulink diagrams prior to translation by the Z Producer in which type information is injected into the model. This can either be done manually, or automatically by means of a typechecker. A separate file is created that contains the type information for blocks, and a tool populates them into the respective Simulink `MDL` file. Type information is made explicit through additional attributes (name/value pairs) in the records that describe information related to the blocks as they are encoded in the `MDL` file of the model.

The above process allows us to make the syntactic matching of the Z Producer, that infers which schemas are used to characterise block functionality, sensitive to (semantic) type information. To explain, the association of entries in the Simulink file and Z schemas is determined by the 'library meta-file' of ClawZ. Fig. 2 shows how we extend the default entries in this file to take into account the injected type attributes. The entries for `Zname` determine the schema used for the block, and the `InputTypes` attributes are additional matching conditions. Instantiation of a block upon translation only takes place if all attributes in the `SelectionParameters` clause are present in the model file. In particular, the `InputTypes` attributes have to exist. Although currently we insert them manually, their generation can potentially be automated by a tool.

In practice, in order to support the new types no low-level modifications to the Z Producer and its code are necessary. The axiomatisation of complex numbers, vectors, matrices, common operators, and Simulink block functionality can be cleanly encapsulated in a collection of designated ProofPower-Z theories. These theories are then merged with the default theory of ClawZ, containing the tool itself as well as the standard library of block definitions. The resulting database of theories is then configured as the default parent when generating the ProofPower-Z database for the specification of particular Simulink models.

```
BlockSpecification {
   Zname Sum_P2
   SelectionParameters {
      BlockType Sum
      Ports [2, 1]
      Inputs "2"
      InputTypes "RR"
  }
}
```

```
BlockSpecification {
   Zname Sum_P2C
   SelectionParameters {
      BlockType Sum
      Ports [2, 1]
      Inputs "2"
      InputTypes "CC"
  }
}
```

Fig. 2. Entries in ClawZ's library meta-file for the Sum block.

3.2 Support for Communication Blocks

The applications we like to consider contain various types of blocks which are initially not supported as part of the translatable subsets of ClawZ. To incorporate support for these blocks we have extended the ClawZ block library. It consists of providing specifications for the blocks as Z schemas and library-meta data to recognise the blocks in Simulink diagrams and thereby generally enable their translation. For reasons of space we cannot comment on all extensions here but will discuss the most significant ones. The complete ProofPower-Z theories are available from http://www.cs.york.ac.uk/circus/tp/tools.html.

Digital Filtering Digital filters are frequently used in signal-processing and communications applications. Their effect can be generally described by the following difference equation.

$$y(n) \; = \; \frac{1}{a_0} \left(\sum_{i=0}^{M} b_i * x(n-i) - \sum_{j=1}^{N} a_j * y(n-j) \right)$$

Here $x(n)$ refers to the input signal and $y(n)$ to the output signal at time step n; the a_j and b_i are the feedback and forward filter coefficients, respectively. Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters are both characterised by the above equation with the difference that for FIR filters N is zero, hence the output only depends on the history of inputs $x(n-k)$ for $k \in 0..M$, and not recursively on previous outputs. As the name suggests, the impulse response function for FIR filters is non-zero only for a finite range whereas for IIR filters it usually extends to infinity.

There are various ways of realising filters by means of primitive elements such as Gain, UnitDelay and Sum blocks realising multiplication with a factor, delay of a signal, and summation. The Direct Form I utilises two cascades of delay blocks as buffers, one for the inputs and one for the output; the Direct Form II, on the other hand, is more compact using only one such cascade. If the filter is modelled by a lower-level subsystem, we can indeed use ClawZ *as is* to generate the Z model — as long as inputs are assumed to be real values.

The Communications Toolbox for Simulink, on the other hand, provides new atomic blocks that support signal filtering. For these blocks we have to provide schema definitions that specify their functionality in order to allow for their

$$
\begin{array}{l}
IIR1 : [a, b : \mathrm{seq}_1\ \mathbb{C}] \rightarrow [x, y : \mathbb{C};\ x_buff, x_buff', y_buff, y_buff' : \mathrm{seq}\ \mathbb{C}] \\
\hline
\forall\, pars : [a, b : \mathrm{seq}_1\ \mathbb{C}] \bullet \\
\quad IIR\ pars = [x, y : \mathbb{C};\ x_buff, x_buff', y_buff, y_buff' : \mathrm{seq}\ \mathbb{C} \mid \\
\quad\quad \#\, x_buff = \#\, pars.b - 1 \land \#\, x_buff' = \#\, x_buff \land \\
\quad\quad \#\, y_buff = \#\, pars.a - 1 \land \#\, y_buff' = \#\, y_buff \land \\
\quad\quad (\exists\, x_sum, y_sum : \mathbb{C} \mid \\
\quad\quad\quad x_sum = SumSeq_C\ (MultSeq\ (pars.b, \langle x \rangle \frown x_buff)) \land \\
\quad\quad\quad y_sum = SumSeq_C\ (\\
\quad\quad\quad\quad \{i : 1..(\#\, pars.a - 1) \bullet i \mapsto pars.a(i+1) *_C y_buff(i)\}) \bullet \\
\quad\quad\quad\quad\quad y = (recip_C\ pars.a(1)) *_C (x_sum -_C y_sum) \\
\quad\quad\quad\quad\quad x_buff' = \langle x \rangle \frown \{i : 1..(\#\, x_buff - 1) \bullet i \mapsto x_buff(i)\} \land \\
\quad\quad\quad\quad\quad y_buff' = \langle y \rangle \frown \{i : 1..(\#\, y_buff - 1) \bullet i \mapsto y_buff(i)\})]
\end{array}
$$

Fig. 3. Schema specifying the digital filter block in the Direct Form I.

translation into Z models. Based on the actual implementation of the filter it is beneficial to use a formal model that is closest to it, we therefore provide a set of (equivalent) specifications reflecting various filter implementations. However, for space considerations we only discuss one of them.

Fig. 3 includes the schema that directly translates the difference equation, resembling the realisation of the filter in the Direct Form I. The schema is obtained by applying $IIR1$ to a binding of type $[a, b : \mathrm{seq}_1\ \mathbb{C}]$, which provides the filter coefficients. Here, a and b are parameters that are extracted from the attributes of the block as it is encoded in the `MDL` file; they are set inside the Simulink tool. The schema has the components x and y, corresponding to the current input and output, and additional state components x_buff and y_buff, including their primed counterparts, to maintain a history of previous inputs and outputs; the latter are modelled by sequences. The length of the buffer sequences is one less the length of the respective coefficient sequences $pars.a$ and $pars.b$ which implicitly determine the dimension of the filter. The two summation terms in the difference equation are assigned to the local constants x_sum and y_sum. Both are used to calculate the output y by multiplication with the reciprocal value of a_0, given by $pars.a(1)$. Observe that $MultSeq_C$ realises element-wise multiplication of complex sequences. Finally it is necessary to shift the contents of the buffers, adding the current input and output as new head elements.

The above schema cannot be directly used by ClawZ since it does not conform to the naming conventions on input and output ports and parameters. We lift it into a block schema $IRR1_Block$ by renaming a and b to $NumCoeffs$ and $DenCoeffs$, as well as x and y to $In1?$ and $Out1!$ to achieve this conformance.

The schema is configured for translation using the library-meta file entry included in Fig. 4. Here, the matching attributes specify `Reference` as the block type, hinting that the block is instantiated from a supplementary Simulink toolbox. It is further classified as a filter by the `SourceType` attribute, and the remaining attributes specify the kind of filter used. Since the block has to transmit several parameters, they also need to be identified in the library meta-file entry; it is done by virtue of the `TransmittedParameters` clause.

```
BlockSpecification {
   Zname IIR1_Block
   SelectionParameters {
     BlockType Reference
     SourceType "Digital Filter"
     TypePopup "IIR (poles & zeros)"
     IIRFiltStruct "Direct form I"
     CoeffSource "Specify via dialog"
   }
   TransmittedParameters {
     NumCoeffs Vector
     DenCoeffs Vector
   }
}
```

Fig. 4. Library meta-file entry for the *IIR*1 filter block.

DFT_Block

$In1?, Out1! : \mathrm{seq}_1\ \mathbb{C}$

$\# Out1! = \# In1? \land (\forall k : 1..(\# In1?) \bullet Out1!(k) =$
$\quad SumSeq_C (\{n : 1..(\# In1?) \bullet n \mapsto In1?(n) *_C$
$\quad\quad exp (\sim_C (0.0, 2.0 *_R \pi) *_C z2c ((k-1) * (n-1)) /_C (z2c\ \# In1?))\}))$

Fig. 5. Block Schema for the Discrete Fourier Transformation.

In addition to the standard filters two instances of adaptive filters have been included, respectively supporting the Least Mean Square (LMS) and Root Mean Square (RMS) algorithms for adjusting the filter coefficients. They are not further discussed here but explained in more detail in [7]. Notably, the specification of the RMS filter required complex matrices to record correlations.

Fourier Transformation Fourier transformations are performed to convert a signal in time into a corresponding signal in the frequency domain. The discrete Fourier transform (DFT) of a signal $x(n)$ for $n \in 0..(N-1)$ and its inverse is defined by the following pair of equations.

$$y(k) = \sum_{n=0}^{N-1} x(n) \exp(-\frac{2\pi ikn}{N}) \quad \text{and} \quad x(k) = \frac{1}{N} \sum_{n=0}^{N-1} y(n) \exp(\frac{2\pi ikn}{N})$$

The Fourier transform $y(n)$ will usually be a vector of complex values that determine the amplitude and phase of equidistant frequencies. The DFT is always an invertible transformation, assuming the signal is repeated periodically.

To support a corresponding block of the Communications Toolbox that performs this operation, we first provide additional definitions for complex exponentiation and the constant π. Exponentiation was defined using the Taylor expansion $e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!}$, but other approaches may be possible too, for example using Euler's equation in the particular case above. This allowed us to define the corresponding block schema for the operation as given in Fig. 5. The function $z2c$ here is a utility operator that converts an integer into a corresponding complex number, and $\sim_C$ is negation of complex numbers.

Modulation The block we will look at in more detail here is the Quadrature Amplitude Modulation (QAM). The essence of QAM modulation is that an

integer value in the range $1..n$ to be encoded is mapped onto a point in the complex plane given by the signal constellation diagram. For 16-QAM ($n = 16$) the latter consists of a regular grid of 16 equidistant points where each point represents one of the symbols in the permissible range 1 to n. The encoding simply involves associating each value with a point in the grid, and the decoding determines the symbol of the point that is closest to a given point represented by the complex (baseband) signal obtained from the demodulated carrier signal.

In order to characterise this functionality as a block schema we first introduce a constant $RQAM_16_SCD : \text{seq}\ \mathbb{C}$ being a sequence that records for each of the values in the range 1 to 16 the corresponding point associated in the complex plane. Encoding is now simply applying the sequence as a function.

For the decoding we have to determine which symbol is closest to the actually received signal. It is realised by the following block schema.

$$
\begin{array}{l}
RQUAM_16_Demod_Block \\
\hline
In1? : \mathbb{C};\ \ Out1! : \mathbb{Z} \\
\hline
Out1! \in 1..16 \land (\forall\, n : 1..16 \bullet \\
\qquad Abs_C\ (In1? -_C RQAM_16_SCD(Out1!)) \leq \\
\qquad Abs_C\ (In1? -_C RQAM_16_SCD(n)))
\end{array}
$$

Above Abs_C is the absolute value of a complex number being its distance from the origin. We simply require that the symbol output by $Out1!$ is the one which has the minimal distance to $In1?$ amongst all points on the grid.

We have encoded several blocks beyond the ones mentioned in this section, for example the Trellis encoding of a signal for a specific convolution code, and its decoding using the Viterbi algorithm. This is reported in more detail in [14].

4 Case Study: Software-Defined Radio

Software-defined radios (SDRs) are radio-communication devices in which components typically found in those devices like, for example, mixers, filters, amplifiers, modulators and demodulators, and so on, are implemented in software rather than being statically realised in hardware. They are designed to sometimes carry out the work of multiple radio devices in a single piece of hardware as they have the ability to support different bandwidths, modulation techniques and communication standards all at once. There are various examples of SDRs in the home and consumer market, for instance in mobile phone devices, but notably military applications profit from their versatility with the addition of encryption and security-related features; this renders the SDR as a potential safety-critical system whose development profits from the use of ClawZ.

The ClawZ extensions we discussed in the previous section have been used to construct the formal model for the Simulink diagram of a specimen SDR which can be found as one of the examples published on-line by MathWorks [5]. The corresponding diagram is included in Fig. 6. It models the modulation and

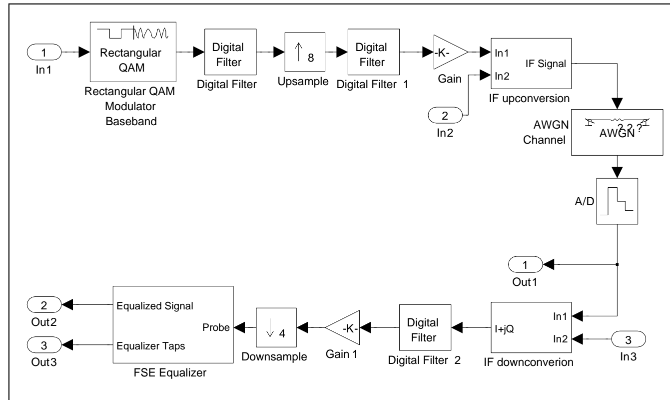


Fig. 6. Control law diagram for a Software-defined Radio

encoding of the source signal, transmission through the ether, and demodulation and decoding of the received signal. The diagram requires the Communications and Signal Processing Toolbox for Simulink. It is hierarchically organised as IF upconversion, IF downconversion and FSE Equalizer are subsystem blocks. (Their respective diagrams are omitted here.) The model is overall not complex in structure, but almost all its basic blocks were taken from the external toolboxes.

The model contains various elements which ClawZ initially did not recognise, and whose support we previously discussed. The input signal In1 is first submitted to a QAM modulator and further passed to a digital filter. These are elementary blocks of the Simulink Communications Toolbox utilised by the model. The following Upsample block changes the rate of the signal, and the output is further submitted to a digital IIR filter. Since the QAM modulation produces a complex output, both filters operate on complex signals. The conversion of the complex (baseband) signal into a real (passband) signal is achieved by the IF upconversion subsystem. It modulates the signal onto a carrier frequency obtained via In2 in order to prepare it for transmission. The transmission of the signal is simulated by a noisy channel, that is the AWGN Channel block.

Upon reception similar operations are performed to the ones already mentioned; we will not explain them in detail here. It shall be noted, however, that the FSE Equalizer subsystem requires the LMS adaptive filtering block.

Prior to the extension of ClawZ very little of the model could actually have been translated. Following the extensions, it was possible, with a few model-specific customisations, to translate the entire diagram, and successfully parse and typecheck the Z specification within ProofPower. A few blocks had to be further added like the AWGN block to simulate additive white Gaussian noise, or the Upsample and Downsample blocks whose function it is to change the rate of a signal. This did not pose a problem in practical terms.

To handle the problem of polymorphic blocks we implemented a Java utility *MdlMergeApp* which merges the attributes of two Simulink `MDL` files. Whereas one file serves as the actually model, the other only contains the residual `MDL`

attributes for types. Keeping the two separate has the advantage that the model can be modified without already specified type information being lost.

We finally shall point out that no verification of code of the SDR diagram has been attempted so far, but the construction of the formal model paves the way for future work on this, including other kinds of formal analysis.

5 Conclusion and Future Work

In this paper we have reported on several extensions to ClawZ that pave the way for its use for generating formal models of control laws typically found in communications and signal-processing devices. It is an application domain that so far could not take advantage of the ClawZ tool support, and our experience suggests it being possible to apply ClawZ for such systems too without incurring fundamental changes to its underlying implementation and architecture.

The extensions entailed the introduction of new data types and additions to the library of supported blocks. To evaluate the additions, we animated respective block schemas using the Z animator Jaza [13,12]. This has been done, where possible, for the formalisation of complex arithmetics, complex vector and matrix operations, and importantly high-level specifications of blocks. In some cases definitions had to be rewritten to be processed by the Jaza tool; for instance, Jaza generally does not allow axiomatic definitions, hence functions such as $IIR1$ had to be rewritten into schemas while setting their parameters to constant values. Further, inductive definitions had to be suitably unfolded.

Further validation took place in the automatic construction of a formal model for the non-trivial control law of a software-defined radio. This case study provided an initial motivation and benchmark for what support may be required for communications applications, but also highlighted imminent problems in using ClawZ in this context. A particular advantage of ClawZ is that it reduces the user interaction in the verification process, and the generation of Z models for diagrams is an aspect which can be entirely automated. On the other hand, the polymorphism of blocks puts limitations on automation in the current translation approach. A solution to this problem is either to alter the translation strategy by formally embedding polymorphic block behaviours, or, as we did, introduce additional steps that inject type information.

Future work consists of first developing a more comprehensive coverage of blocks from the Communications Toolbox of Simulink, and testing our extensions with a larger collection of diagrams of real-world applications. Although our current theories already provide support for many of the blocks, there are, for instance, gaps in supporting the various modulations techniques.

A second major area of follow-up work is to examine the verification of code within the new settings. ClawZ provides a powerful universal proof tactic (Supertac) to discharge verification conditions arising in the verification of code, and a process called 'witnessing' conducts the proof in an incremental manner as to increase the success of the automatic proof steps [2,9]. It is likely that the proof tactic will have to be adjusted to take full advantage of automation.

Finally, a third desirable extension is to implement a tool that automatically injects type information into Simulink models which then, as previously explained, can be exploited by ClawZ in constructing Z models. For this the control law at first must be typechecked, and secondly type information has to be written back into the `MDL` file. We have developed a Java component library that parses and processes `MDL` files, making the latter trivial. The implementation of a typechecker for control laws is pending work, but is not challenging.

Acknowledgements We are grateful to QinetiQ for making their ClawZ tool suite available. Especially, we like to thank Collin O'Halloran and Nick Tudor for their consultation and involvement. We also would like to thank EPSRC for funding this work as part of the research grant EP/E025366/1.

References

1. Lemma 1. ProofPower and ProofPower-Z, 1984–2009. Freely available for download at http://www.lemma-one.com/ProofPower/index/index.html.
2. M. Adams and P. Clayton. ClawZ: Cost-Effective Formal Verification of Control Systems. In *Formal Methods and Software Engineering*, volume 3785 of *Lecture Notes in Computer Science*, pages 465–479. Springer, October 2005.
3. R. Arthan. On Formal Specification of a Proof Tool. In *VDM'91 Formal Software Development Methods*, volume 551 of *Lecture Notes in Computer Science*, pages 356–370, January 1991. Technical report available online at http://www.lemma-one.com/ProofPower/papers.
4. R. Arthan, P. Caseley, C. O'Halloran, and A. Smith. ClawZ: Control laws in Z. In *Third International Conference on Formal Engineering Methods (ICFEM)*, pages 169–176. IEEE Computer Society Digital Library, September 2000.
5. Kelly Bletsis. Software Defined Radio, July 2002. Simulink Model available for download from http://www.mathworks.com/matlabcentral/fileexchange/1987.
6. Public Safety Special Interest Group. Software Defined Radio Technology for Public Safety. Technical report, April 2006. Available online at http://www.ece.vt.edo/swe/chamrad/psi/SDRF-06-A-0001-V0.00.pdf.
7. S. Haykin. *Adaptive Filter Theory*. Prentice Hall Information and System Sciences Series. Prentice Hall, October 2001.
8. F. Jondral. Software-Defined Radio — Basics and Evolution to Cognitive Radio. *Journal of Wireless Communications and Networking*, 2005(4):275–283, 2005.
9. QinetiQ Ltd., 85 Buckingham Gate, London SW1E 6BP, UK. *ClawZ Toolset User Guide*, 2007. Draft document for version 2.2.alpha6 of ClawZ.
10. J. Reeds. *Software Radio: A Modern Approach to Radio Engineering*. Communications Engineering and Emerging Technologies Series. Prentice Hall, May 2002.
11. Inc The MathWorks. Simulink ®, 1994–2008.
12. M. Utting. Data Structures for Z Testing Tools. In *The 4th Workshop on Tools for Systems Design and Verification*, July 2000. Paper is available at http://www.cs.waikato.ac.nz/~marku/jaza.
13. M. Utting. Jaza User Manual and Tutorial, June 2005. http://www.cs.waikato.ac.nz/~marku/jaza/userman.pdf.
14. M. Vernon. The Modelling and Verification of Software-Defined Radio Techniques in Communication Applications. Master's thesis, University of York, Heslington, York, YO10 5DD, UK, May 2008.