# The SKA Costing and Design Tool

Dominic Ford[1], Rosie Bolton[1], Tim Colegate[2], Paul Alexander[1] and Peter Hall[2].

[1] *Cavendish Laboratory, J.J. Thomson Avenue, Cambridge, CB3 0HE, UK.*
[2] *Curtin Institute of Radio Astronomy, Perth, Western Australia.*

20th January 2009

**Abstract**

The SKA Costing and Design Tool provides a framework in which hierarchical descriptions of telescope designs can be built and costed as a function of input parameters and telescope performance. The tool allows engineers and astronomers to rapidly explore the possible parameter space of SKA designs, probe the cost vs. performance tradeoffs which affect them, and ultimately produce optimised designs for the SKA. In this memo, we describe how the costing tool works and show screenshots of its graphical user interface.

## Contents

# 1 Introduction

Intrinsic to the design of any telescope are trade-offs between cost and scientific performance (see, e.g., Gaensler and Lazio, 2006). In the design of the Square Kilometre Array (SKA), these trade-offs will be unusually complex, for two reasons. Firstly, its design will have an unusually large number of free parameters: there will be many design choices to be made in addition to the familiar considerations of sensitivity and survey speed. For example, it will use a hybrid of detector technologies co-existing side-by-side – both the International SKA Project Office[1] (ISPO) Reference Design (ISPO, 2006; Schilizzi *et al.*, 2007) and the SKADS Benchmark Scenario (Alexander *et al.*, 2007; Bolton *et al.*, 2008) favour a three-component hybrid of phased aperture arrays, dishes with wide-band feeds and either close-packed aperture arrays or phased-array feeds (PAFs) – and the frequency coverage and collecting area of each component must be decided upon. Within the back-end processing for the aperture arrays, the choice between digital and analogue beam-forming remains to be made; Alexander *et al.* (2007) argue that even if one becomes the favoured option, the other should be developed in parallel for the foreseeable future to provide a realistic fallback alternative. Within the dish-based collectors, issues such as the dish size and feed type remain to be finalised.

The second source of complexity in the SKA's cost-performance trade-offs is the multi-faceted nature of its science programme (see, e.g., Gaensler, 2004; Jones, 2004). Its prospective users each place different requirements upon the SKA's angular resolution, survey speed, sensitivity and frequency coverage, and a compromise will have to be reached (see, e.g., Jackson, 2003, 2006).

Whilst the ISPO Reference Design and the SKADS Benchmark Scenario documents have presented and discussed in detail a range of manually-optimised skeletal designs for the SKA, it is clear that more immediate access to cost estimates and scientific simulations of telescope designs will be essential before engineers can make significant progress in charting the available parameter space. To this end, the SKA Costing Tool has been developed on behalf of the SKA Program Development Office (SPDO) and in part as work package DS3-T3 of the SKADS programme. This tool acts upon telescope designs which are described in a hierarchical fashion, in which large *design blocks* – for example, the whole SKA – subdivide into smaller units – for example, SKA Stations – until eventually the hierarchy reaches *components* – indivisible atomic units. One possible view of how this hierarchy might appear is shown in Figure 1. The tool can calculate the number of components required to build any given design block, propagate costs through this

[1]The International SKA Project Office is now the SKA Program Development Office (SPDO).

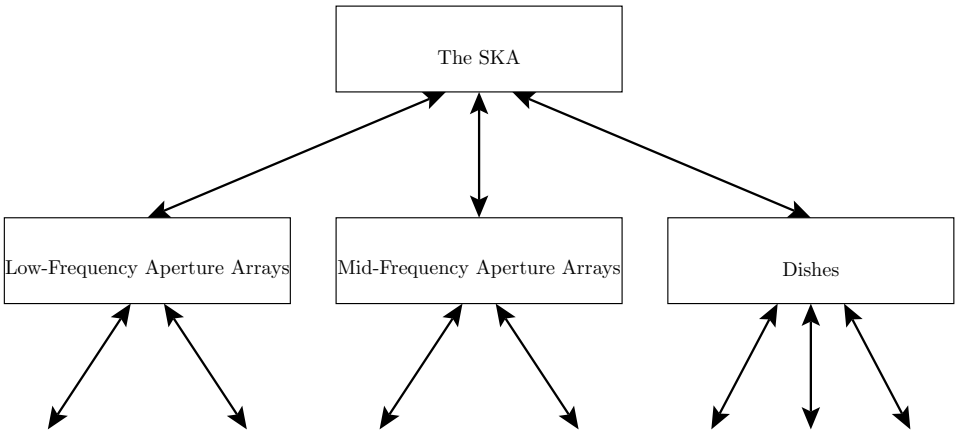


Figure 1: A simplified example of how the top of a hierarchical telescope design might appear as a network of design blocks; this example is based loosely upon the structure of the SKADS Benchmark Scenario (Alexander *et al.*, 2007).

hierarchy to sum up to total cost of any given telescope design, and provide a breakdown of the contributions of each design block to the total cost. In addition, the tool can also propagate other arbitrary quantities through the hierarchy, including, for example, the power consumption of components and their data rates. The former is useful for calculating the total power requirements of the SKA, meanwhile the latter is an unusual case, where data processing steps can *reduce* the total data flow to their parents.

A key feature of the Costing Tool is that the hierarchical telescope designs which it acts upon are *scalable*; they do not represent a single telescope built to a particular specification, but rather take a wide range of input parameters, and within reasonable limits, can estimate what components would be required to build a telescope to any requested specification. This does not replace the need for specialist engineers: the calculations performed by the tool are *estimates* based upon simple models, which would need to be extensively expanded upon before they could actually be built. In the first instance, our aim is to produce realistic models of how the cost of the SKA will *scale* with design parameters, not to produce accurate absolute costings.

The Costing Tool builds upon the work of its predecessor, *SKAcost*, which was developed by the Australia Telescope National Facility (ATNF), with some financial support and system engineering input from the ISPO. *SKAcost*, which itself used the modular approach first described in Horiuchi *et al.* (2004), was also a hierarchical tool which could estimate component costs as a function of purchase date for dish-based SKA designs (Chippendale *et al.*, 2007; Schilizzi *et al.*, 2007). In addition, a variant version of *SKA-*

*cost* has been developed by the ATNF for their studies of the cost vs. performance trade-offs for the Australian Square Kilometre Array Pathfinder (ASKAP). *SKAcost* was written in Python and two user interfaces were developed for it: experienced Python programmers were encouraged to modify parameters directly within its Python code, giving them essentially unlimited scope to explore parameter space, meanwhile an online CGI-based web interface provided a more user-friendly way to access it, although only a selected subset of parameters could be varied using it. The tool provided uncertainty analyses based on simple statistical modelling and reports in a variety of financial formats, including Net Present Value summaries.

The new Costing Tool greatly improves upon the user-friendliness of *SKAcost* and seeks to make its calculations as transparent as possible. It aims to provide all of the flexibility which was available to *SKAcost*'s technically-minded users, in an interface which is at least as user-friendly as *SKAcost*'s web-based interface. Within the Costing Tool, the hierarchical structure of telescope designs can be interrogated in a graphical environment, as can the cost breakdown between the various parts of the hierarchy. Telescope designs can also be modified in this graphical environment, and, in the case of simple modifications, no programming experience or knowledge of Python is required. More complex modifications, including the building of entirely new telescope designs, are possible with a basic knowledge of Python: small fragments of Python code need to be written to instruct design blocks how to calculate their resource requirements from any given set of inputs. Significant effort has been put into making this process as user-friendly as possible. The development of the Costing Tool is ongoing and will continue throughout the PrepSKA programme, during which time it will be extended and coupled to technical simulations of the SKA.

## 2 The Structure of the Costing Tool

Within the Costing Tool, we maintain a sharp division between the *costing engine* – the software used to calculate the numbers of components needed to build an SKA to a particular specification – and the *telescope designs* which it acts upon. This separation ensures that telescope designs can be modified by engineers without exposure to the lower-level Python implementation of the cost calculations; it also ensures that cost calculations are performed in a homogeneous way between all design blocks, since they must all use the same costing routines. In addition, the costing engine is separated from the user interfaces which it uses to communicate with the user and with the outside world. This structure is illustrated in Figure 2.

Three user interfaces to the Costing Tool are currently available. It is anticipated that most users will use the graphical interface on account of its user-friendliness. Some users with scripting experience may prefer the

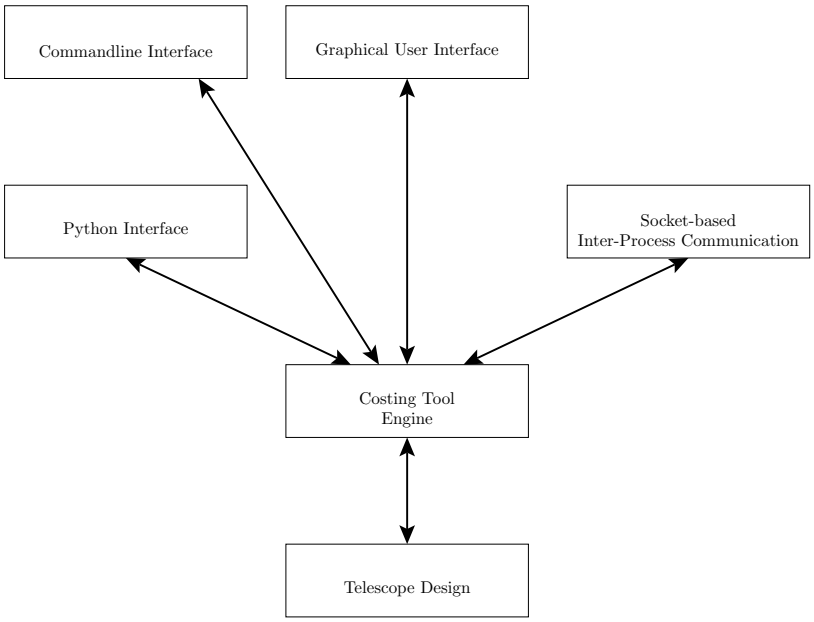


Figure 2: A diagram of the modular structure of the Costing Tool. The costing engine is separated both from the telescope design models it acts upon, and from the interfaces with which it communicates with the user and with other software packages. This means that the user is able to modify telescope designs without detailed knowledge of how the costing engine works, and that experienced programmers can use the Costing Tool from within the programming language of their choice.

commandline interface, which provides a similar degree of flexibility to the graphical interface. Finally, more experienced programmers may wish to use the Costing Tool directly from within Python, where they will be able to develop their own independent user interfaces to the tool, specialised for their own particular requirements. In view of the sophistication of the new tool, there is currently no web-based interface to match that which was available for *SKAcost*, and we envisage that users will run the graphical user interface on their own local machines.[2]

As Figure 2 indicates, the Costing Tool can also be accessed by other applications via socket-based inter-process communication, which means that other design optimisation tools can query live cost estimates for designs from the tool. As a proof of this concept, a SKADS tool is under development which will optimise the cable and trenching layouts between SKA stations. It will communicate with the Costing Tool to obtain information about the fibre cost model, and will communicate results back into the cost calculations.

As well as the division which we draw between telescope designs, the Costing Tool engine and its user interfaces, we also divide up telescope designs into separate sharply-divided blocks. Each design block within a telescope design hierarchy is a *black box*: it takes a defined set of inputs from its *parent* design blocks, operates on them in some way which other design blocks need not be concerned with, and then returns a defined set of outputs. This information flow is illustrated in Figure 3. This approach means that the various design blocks in a telescope design can be efficiently written by a variety of different engineers working at different institutions, each bringing their own specialist expertise to their own particular areas. Because each block is a black box, each can straightforwardly be re-implemented in the light of new information or expertise, without changing the rest of the telescope design around it. Within the Costing Tool, it is straightforward to swap one design block for another within a telescope design, to obtain a rapid comparison of the costs calculated by different models.

In addition, the Costing Tool makes it easy to re-use components and design blocks in many different places in a hierarchical telescope design if they are designed to take a sufficiently general set of inputs that they can be used in several different situations.

# 3 Cost Modelling

The Costing Tool has a variety of features for modelling the way in which costs scale with purchase date and for modelling the uncertainties in the prices of goods.

[2]The Costing Tool is known to run under Microsoft Windows, Mac OS X and Linux. It is believed to run under all other POSIX-compliant operating systems.

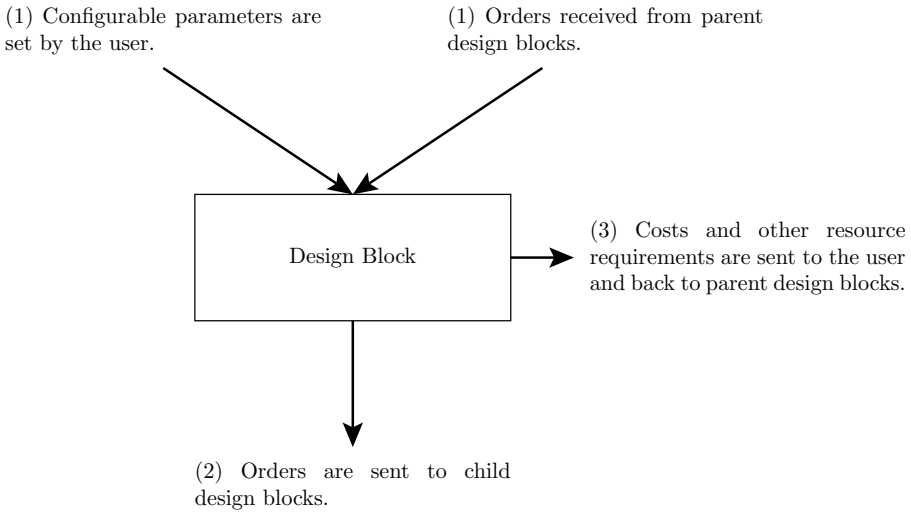


Figure 3: A diagram of the data flow which takes place within design blocks. First of all, in Step (1), the design block receives *orders* from all of the parent design blocks which make use of it, together with some user-configurable parameters. The design block then, in Step (2), computes how many child design blocks are needed to fulfill these orders, sends orders through to child design blocks, and calculates any additional "glue" costs which are associated with putting these sub-components together. Finally, the design block returns its total resource requirements – not just cost, but also power consumption and any other arbitrary attributes which the user wishes to propagate through the hierarchy – to the user and to its parent design blocks.

## 3.1 Cost Scaling

In general, the costs of telescope components vary over time: for example, the prices of support materials may rise following inflation, meanwhile the prices of computational hardware may fall following Moore's Law. Since the SKA will be built over an extended period of time, with investment in different parts of the telescope infrastructure at different times, it is important to take account of these scalings, and we now describe how the Costing Tool models them.

For each component, an estimated *reference cost* $c_0$ is specified, together with the year $t_0$ for which this cost estimate is being made. An indication of the uncertainty in the cost is also specified: for example, this could be a statement that the cost lies somewhere between two limits, but that no more accurate information is available, or it could be a statement that the cost has a certain mean and standard deviation. In general, the user can state that the probability distribution function of the cost has one of four possible shapes, as shown in Figure 4. Wherever possible, these costs are accompanied by references, to give an indication of their source and degree of reliability. In some cases, the reference may merely state that the cost given is a guess; ideally they will take the form of quotes from manufacturers.

These reference costs are accompanied by an estimate of how they are expected to scale with time. In many cases, the expectation will be that they will follow either inflation or Moore's Law, both of which are preprogrammed as power laws into the Costing Tool, although the user is free to define his own scaling relations. When a telescope design is found to require the purchase of a component in some particular year $t_1$, the reference cost is scaled from the year $t_0$ to $t_1$ using the stated scaling relation.

One key benefit of the separation of telescope designs from the costing engine within our tool is that cost scalings are performed in a homogeneous fashion across all parts of a telescope design. As an example of the practical use of this, if the user wanted to assess how sensitive his cost estimates were to the continuation of Moore's Law over the next decade, then he could do so by changing the power-law slope of Moore's Law within the costing engine to represent a more pessimistic forecast. This would affect all components which depended upon Moore's Law in a homogeneous and self-consistent fashion.

## 3.2 Net Present Values

In addition to the change in the prices of components over time, currency units also change in value: purchases which cost the same number of euros, but in different years, do not represent the same amount of material expenditure. To first approximation, credit crunches being neglected, currencies devalue in line with inflation.

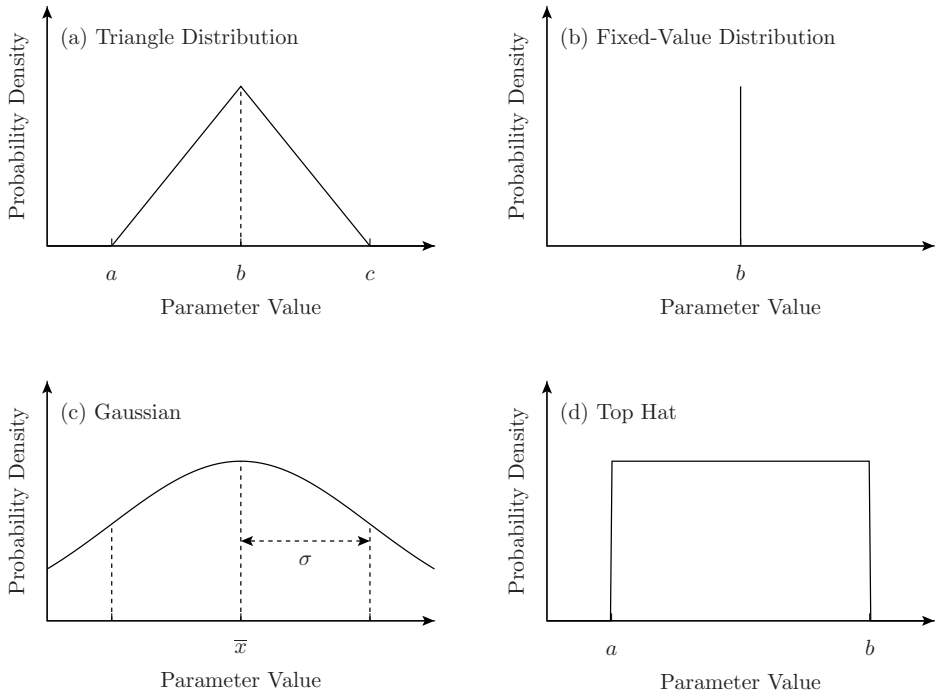


Figure 4: Components include an indication of the uncertainties in their costs. The user has a choice between (a) stating that the component's cost lies somewhere between a pair of lower, $a$, and upper, $b$, limits, and most probably around $c$; (b) stating that there is negligible uncertainty in the cost; (c) stating that the cost has a known mean $\overline{x}$ and standard deviation $\sigma$; or (d) stating that the cost could lie anywhere between a pair of lower, $a$, and upper, $b$, limits. For each case, the probability distribution function for the cost is shown.

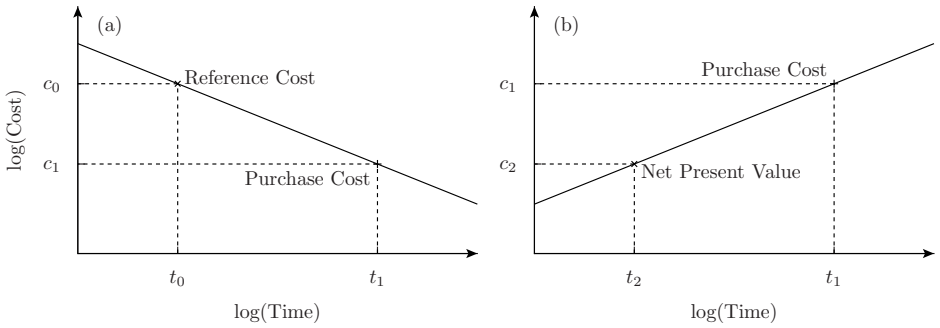


Figure 5: A schematic view of how the Costing Tool models the time evolution of component costs. For each cost component, an estimated reference cost $c_0$ is supplied for a reference year $t_0$, together with information about the uncertainty in this estimate. Ideally this reference cost will be based upon a quote from a manufacturer. Additionally, some indication is supplied of how this cost is predicted to scale with time. When this component is purchased in some chosen build year $t_1$, this scaling law is used to estimate the cost $c_1$ of the component in that year, as shown in Panel (a). Finally, if the user chooses, all purchase costs are then reported as net present values $c_2$ in a given reference year $t_2$ – that is, the amount of money which is needed in a bank account in that reference year to make the required purchase at a later date, taking into account the interest paid on money invested in the bank. To do this, all purchase costs are scaled back to this reference year using the discount rate, as shown in Panel (b).

At present, the Costing Tool offers the capability to report all costs as *net present values* (NPVs), which are defined to be the amount of money which would hypothetically need to be held in some given reference year, such that a telescope could be built over an extended period of time, assuming that those funds which have not yet been spent are invested and accrue interest until they are needed. In Figure 5, we show how the costs of the components which make up the SKA are converted into NPVs. This matches the accounting procedures followed by *SKAcost*, and allows for an easy comparison between the output of the two tools.

However, because NPVs do not represent the way in which funding bodies are likely to allocate resources, we will provide a range of accounting procedures in future versions of the tool. In the meantime, we offer a single alternative, which is to simply return costs as summed expenditure.

### 3.3 Modelling Uncertainties in Cost

Within telescope designs, each individual component stores information about the degree of uncertainty in its cost and other resource requirements.

Central to the Costing Tool is the ability to assess how these uncertainties propagate through into the final cost of the SKA.

To obtain a single cost estimate for the SKA, or for a design block, a single sample is taken from the probability distribution for the cost of each of its constituent components. These samples are taken in such a way that if one particular component appears many times in a hierarchy, the same sample is used every time, since these purchase costs are not independent random variables. To build up a picture of the probability distribution for the cost of the SKA, this process is repeated many times, and the samples used to construct a histogram. This approach has the strength that it makes no assumptions about the forms of the probability distributions for the costs of individual components. While the standard error combination formula[3] provides a faster way of propagating uncertainties, it assumes that the probability distributions are all Gaussian. The central limit theorem states that this will be the case when many independent probability distributions are added together, but this is not necessarily true of the cost of the SKA, which may be dominated by the purchase of large quantities of a small number of different components.

The resulting probability distribution for the total cost of a design block can be returned to the user as either a mean and standard deviation, or as a full plotted histogram.

# 4 The Structure of Telescope Designs

In Section 1 we outlined how telescope designs break down into a hierarchy of design blocks and components. In this section, we outline in more detail the data which is stored in each of these components and design blocks, and how they interact with one another.

## 4.1 Components

We use the term *component* to refer to the atomic elements in telescope designs which do not break down into any smaller sub-components; they form the leaf nodes of the hierarchy. They usually represent material commodities which can be purchased from a supplier, but may also be used to represent expenses which are harder to quantify such as that of a man-hour of manual labour. In simplistic telescope designs, components may also be used to assign pre-determined place-holder costs to particular parts of the telescope design before they are modelled in any further detail. Components contain the following information:

[3] That is, adding all of the uncertainties in the cost components in quadrature.

### 4.1.1 Costs

Each component has two costs associated with it. The first is a one-off *development cost*, which is incurred only once, if the component is used anywhere in a telescope design. The latter is a per-unit manufacture cost, called its *unit cost*, which is incurred every time it is built. As described in Section 3.1, each of these is specified as a reference cost, estimated for a specified reference year, and must be accompanied by an estimate of how the cost will evolve with time.

### 4.1.2 Power Consumption

It is also possible to specify the power consumption of a component, together with its uncertainty, allowing the Costing Tool to estimate the power consumption of SKA designs and of each of their constituent stations. In the future, we will develop telescope designs which feed this data back into their cost calculations to give an indication of the power infrastructure and component cooling that they require.

### 4.1.3 Textual Documentation

In addition to providing numerical data, the authors of components are expected to provide textual documentation to make clear what their components represent physically. They are expected to provide an indication of the reliability of the prices that they quote – for example, whether they are guesses or whether they are based on quotes. They are also encouraged to provide, insofar as they are not commercially sensitive, direct references to quotes from manufacturers, together with an indication of the current state of development of the technology being proposed for purchase. One indication which component authors may use is the ten-point *technology readiness level* (TRL) scale developed by NASA, which indicates the state of a research and development (R&D) programme – whether it is a product which is already shipping, a product which has been prototyped, or whether it is yet to even reach that stage of development.

In order that users may know where to direct enquiries about the components used in telescope designs, authors of components are also required to provide their name and contact details.

### 4.1.4 Specifications

Machine-readable meta-data may be attached to components in the form of *specifications*. Components may have any number of specifications, each having a unique name, a value, and optionally, a physical unit. Parent design blocks can query the specifications of child components in the course of

ordering them, and make use of the values in their calculations. Specifications can contain any information which is relevant to the component, and may take the form of numerical values which are intended to be machine-read by parent design blocks, or take the form of textual information which is intended purely for human readers of the component definition.

## 4.2 Design Blocks

We use the term *design block* to refer to elements of hierarchical telescope designs which are larger than components, and which are made up of smaller parts which are modelled by other components and design blocks.

### 4.2.1 Simple Shopping Baskets

To illustrate what design blocks are, we begin by considering a simple toy example, which we will call *CabledTrench*. This represents a communications link which requires a trench to be dug and five cables to be laid into it. We will measure the cost of this design block per unit metre; for each metre that is purchased, it needs to order one metre of trench and five metres of fibre.[4] We term this kind of design block a *shopping basket*: it doesn't perform any detailed calculations; it simply groups a set of items together. This grouping together is achieved by specifying that the design block has two *children* – a term that we use to describe the sub-components and sub-design-blocks which need to be purchased in order to put a design block together. Our example design block might contain the following information about its children:

Child 1
**ImplementationDB:** Comms_A_Trench
**UnitQuantity:** 1

Child 2
**ImplementationDB:** Comms_A1_Fibre
**UnitQuantity:** 5

The field *UnitQuantity* specifies the number of units of each child which should be ordered for each unit of *CabledTrench* which is bought. The field *ImplementationDB* specifies the name of the child design block or component which is to be bought in each case.

The four pieces of information above represent essentially all that is required to build a shopping basket design block, although, as with compo-

[4]For the purposes of this toy example, we will neglect considerations such as the need for repeater stations, etc.

nents, we also expect design block authors to provide some textual information about who they are and what their design blocks physically represent.

### 4.2.2 More Advanced Design Blocks

Many design blocks can be modelled as simple groupings of items to make shopping baskets which can be ordered just like components, by specifying a quantity and a purchase year. However, in order to make a useful telescope design it will be necessary at some point to write some high-level design blocks which take more physically interesting inputs of the kind that users are apt to supply – for example, system temperatures and desired bandwidths and sensitivities. These design blocks must implement models to convert these inputs into quantities of parts which need to be ordered.

To write design blocks which are more complicated than the shopping baskets met in the previous section, it is necessary to write a few lines of Python code to perform the calculations required to implement such models. In this section we will give a flavour of this process, to show that only a basic knowledge of programming is required to implement simple models. We will assume that the reader has a *basic* understanding of Python syntax, or else has Guido van Rossum's online *Tutorial* (van Rossum, 2008) to hand; users who are not interested in these details may safely skip onto the next section. This memo is not the place for a complete tutorial in how to write design blocks; readers looking for a detailed guide should turn to the Costing Tool Users' Guide (Ford, 2009).

From the point of view of a design block author, the Costing Tool can be seen as a mechanism for passing messages between design blocks. When one design block orders another, a *message* is passed from the buyer to the block which is being bought. In the previous section, this message passing was taking place behind the scenes: one design block could specify that it needed some others to be bought, and the purchasing happened invisibly without any further intervention from the user.

Messages take the form of Python dictionaries. Simple orders, of the kind being generated by the shopping baskets in the previous section, take the form of messages containing only two fields – the number of units to be purchased, and the year in which they are to be purchased, for example:

```
message = {'quantity'     : 1,
           'purchase_year': 2012
          }
```

Shopping baskets and components only accept messages containing these two inputs; if they receive a message containing additional fields they return an error to alert the user that they are not able to understand the information supplied to them. When more complicated design blocks are written, it

is possible to specify a list of the inputs which they are permitted to receive, and the range of values which each are allowed to take.

Each time a design block receives a message, the tool first checks that all of the message's fields are valid inputs to the design block, and then it executes a Python code fragment which is supplied as a part of the design block's definition[5], passing it the received message. The code fragment has access to various function calls which allow it to order child blocks and access other core tools within the costing engine. If necessary, the code fragment is also allowed to use the Python `import` command to make use of any of the standard Python modules. As the Python developers have included a rich array of plug-in modules as standard with the language, many common tasks can be automated with library routines. For example, the `re` module can be imported to provide an efficient implementation of Regular Expressions, or the `pyfits` module can be imported to provide a reader for datafiles in `fits` format.

In this document we will look at only one of the functions to which these code fragments have access: the `SendMessage()` function, which is used to send orders to child design blocks. Its calling syntax is:

```
SendMessage(Name, quantity, year, ExtraInputs={})
```

The first argument, `Name`, should be set to the name of the design block to which the message should be sent. To send a straightforward order to a component or shopping basket, the fields `quantity` and `year` should be set respectively to the number of units of the child to be ordered, and the year in which they should be bought. The field `ExtraInputs` is only used when ordering child design blocks which take their own special inputs: in this case, the additional data fields which are to be sent to the child should be placed in this dictionary.

As an example, the following is a null code fragment: it implements a shopping basket which behaves in the same way as if no Python code fragment had been supplied at all. We show it, despite its uselessness, to demonstrate how few lines of Python code need to be written to make simple interactions with the costing engine:

```
1  for ChildName in ListOfChildren.keys():
2    SendMessage(ChildName,message['quantity'],
                 message['purchase_year'])
3  outputs['NumberOfUnits'] += message['quantity']
```

Line 1 starts a loop over all of the design block's children: `ListOfChildren` is a dictionary which the costing tool passes to the code fragment, whose keys are the names of each of the design block's children. On each iteration

[5]If no code fragment is supplied, then the design block acts as a shopping basket by default.

of this loop, the variable `ChildName` takes one of these names in turn. In Line 2, we send an order to each of these children, reading the number of units to be bought and the purchase year from the dictionary `message` sent to the design block from its parents.

Finally, in Line 3, we update the dictionary `outputs` which the code fragment uses to return output information back to the costing tool. In this case we update a field called `NumberOfUnits` to count the total number of purchases of this design block which have been made.

As an example of a more typical design block code fragment, the following implements a simple model of a close-packed array of aperture array antennae:

```
 1  spacing  = message['Antenna_Spacing']
 2  height   = message['Aspect_Ratio'   ] * spacing
 3  quantity = message['quantity'       ]
 4  date     = message['purchase_year'  ]
 5  SendMessage('LNA'             ,quantity, date)
 6  SendMessage('Filter_Regulator',quantity, date)
 7  SendMessage('Antenna_Passives',quantity, date)
 8  SendMessage('Feed_Board'      ,quantity, date)
 9  SendMessage('element_material',spacing*height*quantity,
                date)
10  SendMessage('Ground_Plane',spacing*spacing*quantity,date)
11  outputs['NumberOfUnits'] += quantity
```

This design block takes four inputs: the number of antennae in the array (`quantity`), the spacing between them, the ratio of their height to their spacing (`Aspect_Ratio`), and the year in which they are to be bought (`purchase_year`). Each antenna element requires the purchase of a low noise amplifier (`LNA`), a filter/regulator, some passives and a feed board (see Lines 5–8). The total area of element material required is the product of the antenna spacing, the height of each antenna, and the total number of antennae (see Line 9). The total amount of ground place material required is the product of the number of antennae and the square of the antenna spacing (see Line 10).

## 4.3 Telescope Designs

Telescope designs bring together collections of components and design blocks to make complete telescope systems. Each contains a list of the names of all of the blocks which comprise the telescope design, the name of the top-level block which represents the whole telescope, and the values of all of the inputs which should be passed to this top-level design block by default.

They also contain a *system design block database* (SDBD), which we have introduced to make it as easy as possible to swap design blocks for

alternative implementations. To illustrate the need for this, we return to the *CabledTrench* example which we described in Section 4.2.1. Imagine that a colleague were to pass you a design block called *CabledTrench_Bob*. The first thing that you would be likely to want to do would be to plug Bob's design block into a standard telescope design in place of the old *CabledTrench* design block, to see how it changed matters.

However, you *wouldn't* want to have to change every reference to *CabledTrench* to *CabledTrench_Bob* in the telescope design. The SDBD is the mechanism we use to make such changes easy. When this design block is ordered, the purchase is propagated through to the *Trench* child design block. However, the tool does not immediately go off and look for a design block on disk called *Trench*. First of all, it looks in the design block's list of children, where it finds a mapping from the *local name* of *Trench* – used internally within the design block – to an *implementation name*. By switching the implementation name in the design block's list of children, we can make it look for different children without modifying its Python code fragment, which exclusively refers to children by their local names.

A second layer of dereferencing occurs in the SDBD. The implementation name of the desired child is then mapped by SDBD to the name of a design block on disk. By changing the relevant entry in the SDBD, we can make all design blocks with children of a common implementation name read a different design block from disk. This architecture is illustrated in Figure 6.

# 5 The Costing Tool Example Telescope

To demonstrate the use of the Costing Tool in practice, it is necessary to have a telescope design for it to act upon. The process of putting together complete and realistic telescope designs in the tool will require extensive consultation with expert engineers throughout the SKA community, which we have not yet begun. In the meantime, we have developed an *Example Telescope*, which is loosely based upon the Benchmark Scenario from the Second SKADS Design and Costing Memo (Bolton *et al.*, 2008). This design is not intended to serve any purpose other than as an example, and is not presented as a fully-costed SKA design. The presentation of plausible costed telescope designs built using the Costing Tool is deferred for future memos.

For reference, the Example Telescope consists of 250 mid-frequency aperture array stations, each 24 m in radius, 250 low-frequency aperture array stations, each 82 m in radius, and 2,480 dishes, each 15 m in diameter. It lacks any model of the correlation or computational hardware, for which arbitrary placeholder costs are used.

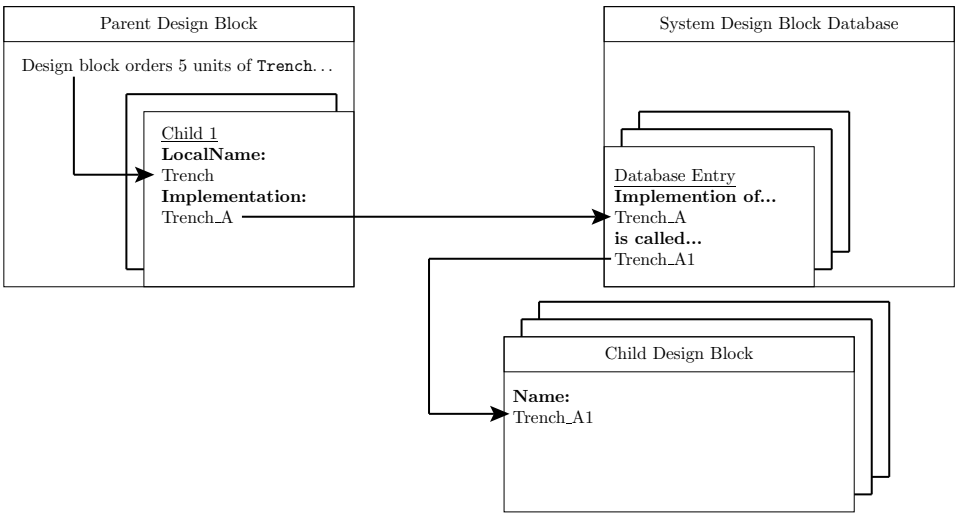


Figure 6: A diagram of the mapping of design block names, using the example design block *CabledTrench* from Section 4.2.1. Within parent design blocks, child blocks are referred to by *local names* – in this case, *Trench*. The design block's list of children map each local name to an *implementation name*. These are in turn mapped to the names of design blocks on disk by the telescope design's *system design block database*.

# 6 The Graphical User Interface

In this section, we use the Example Telescope to demonstrate the Costing Tool's graphical user interface. To emphasise that the Example Telescope is not a complete design for the SKA, and that its cost does not represent a good cost estimate for the SKA, we have chosen to display all design block costs in a ficticious currency which we have called the SKA Accounting Unit (SAU); its value roughly matches the cost of the whole SKA.

When a telescope design is opened within the graphical interface, all of the components and design blocks which make it up are listed in a spreadsheet, as shown in Figure 7. Once the telescope has been built, this spreadsheet also displays the quantities of each block which were required to build the telescope in the chosen configuration. The column *cumulative cost* shows the total amount spent on each block, including the cost of any child blocks further down the hierarchy which were required to make them up. The column *intrinsic cost* is similar, but does not include the cost of the child blocks, and hence it shows only the costs which are uniquely associated with each block. The sum of all of the values in this column equals the total cost of the telescope design – in this case, 0.887 SAU. This spreadsheet can be exported to a CSV file for examination in external software packages.

The graphical interface also allows components and design blocks to be

browsed in a hierarchical display, giving a rapid visual sense of how they fit together. The hierarchy of blocks which make up a low-frequency aperture array station in the Example Telescope is shown in Figure 8. The user is able to click on blocks in the hierarchy to display information about them or to navigate around the telescope design, expanding and collapsing nodes along the way.

In Figure 9, we have gone back to a spreadsheet view, and chosen to display information about the top-level SKA design block. In keeping with the notation above, the panel *total cumulative cost of design block* shows the total cost of the selected design block, including money spent on child blocks, meanwhile the panel *total intrinsic cost of design block* shows only the costs which are uniquely associated with it. The latter can be thought of as the *glue* cost associated with putting the child blocks together. Below this panel, the *cost break-down* box shows what percentage of the cumulative cost of the design block is spent on each of its child blocks.

At the bottom of the display is an indicator to show the details of the telescope system which is currently built, for which costs are currently being displayed. In Figure 10 we expand this panel to show some of the top-level input parameters which the Example Telescope takes.

In all of the panels shown so far, quantities with uncertainties in their values have been displayed with stated standard deviations. However, it is also possible to plot full probability distribution functions (PDFs) for any of these values, to check for non-Gaussianity in their probability spread. These plots are produced via a Monte Carlo approach: many random samples are taken and put into bins. A smoothed cubic spline is fitted through the probability densities calculated for each of the bins to aid the eye in following the trend. In Figure 11 show an example PDF for the total cost of the Example Telescope. The black crosses are the probability densities calculated for each of the individual bins, and the grey curve is the smoothed fit through the noisy datapoints.

In Figure 12 we demonstrate the strength of using scalable telescope designs in the tool – i.e. telescope designs which take many input parameters and which can build telescopes to a range of specifications using simple scaling models. In this simple example, we request the tool to make a one-dimensional parameter survey of how the cost of the SKA scales with the desired sensitivity of the dish component of the array, by building 12 sample telescopes along a line in phase space. On this plot, the horizontal axis measures the requested sensitivity of the dishes, in units of $m^2/K$, meanwhile the vertical axis measures the most likely cost of the SKA. The cost of the SKA rises when the desired sensitivity is increased because a larger collecting area is required, which in turn means that more dishes are required.

In Figure 13, we show the graphical interface's editor for changing component and design block definitions.

# 7 Summary

We have constructed a tool which allows the user to construct hierachical telescope designs in a graphical environment, and which can then propagate costs, power consumption, data rates, and other arbitrary quantities through these hierarchies to evaluate the total cost of any given telescope. In due course, we envisage that we and the SKA Program Development Office (SPDO) will work in collaboration with specialist engineers to produce a range of reference telescope designs within this framework. Over the course of the PrepSKA programme, we also plan to link the Costing Tool into technical simulations of the SKA's science performance.

This work is supported by the European Community Framework Programme 6, Square Kilometre Array Design Studies (SKADS), contract number 011938.

Figure 7: A spreadsheet of all of the components and design blocks in the Example Telescope Design. In this figure, we have chosen to sort the blocks in order of their cumulative cost, so that the most expensive block, the whole telescope, appears at the top. Further details are in the text.

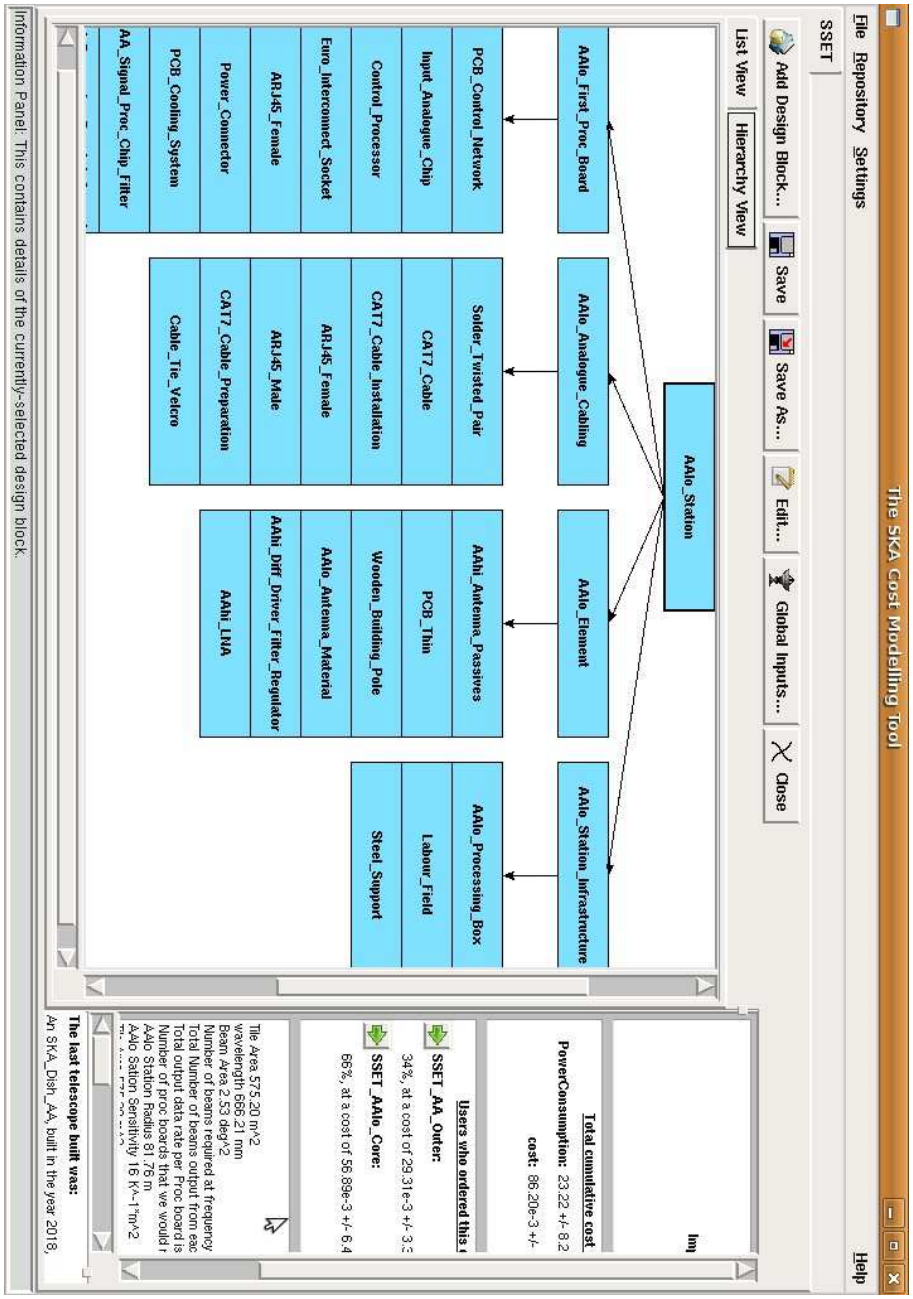


Figure 8: The hierarchy of design blocks which are required to build an *AAlo_Station*. Where strings of blocks appear in vertical lists, all of the blocks in the list are children of a common parent above, but they are too numerous to be displayed horizontally.

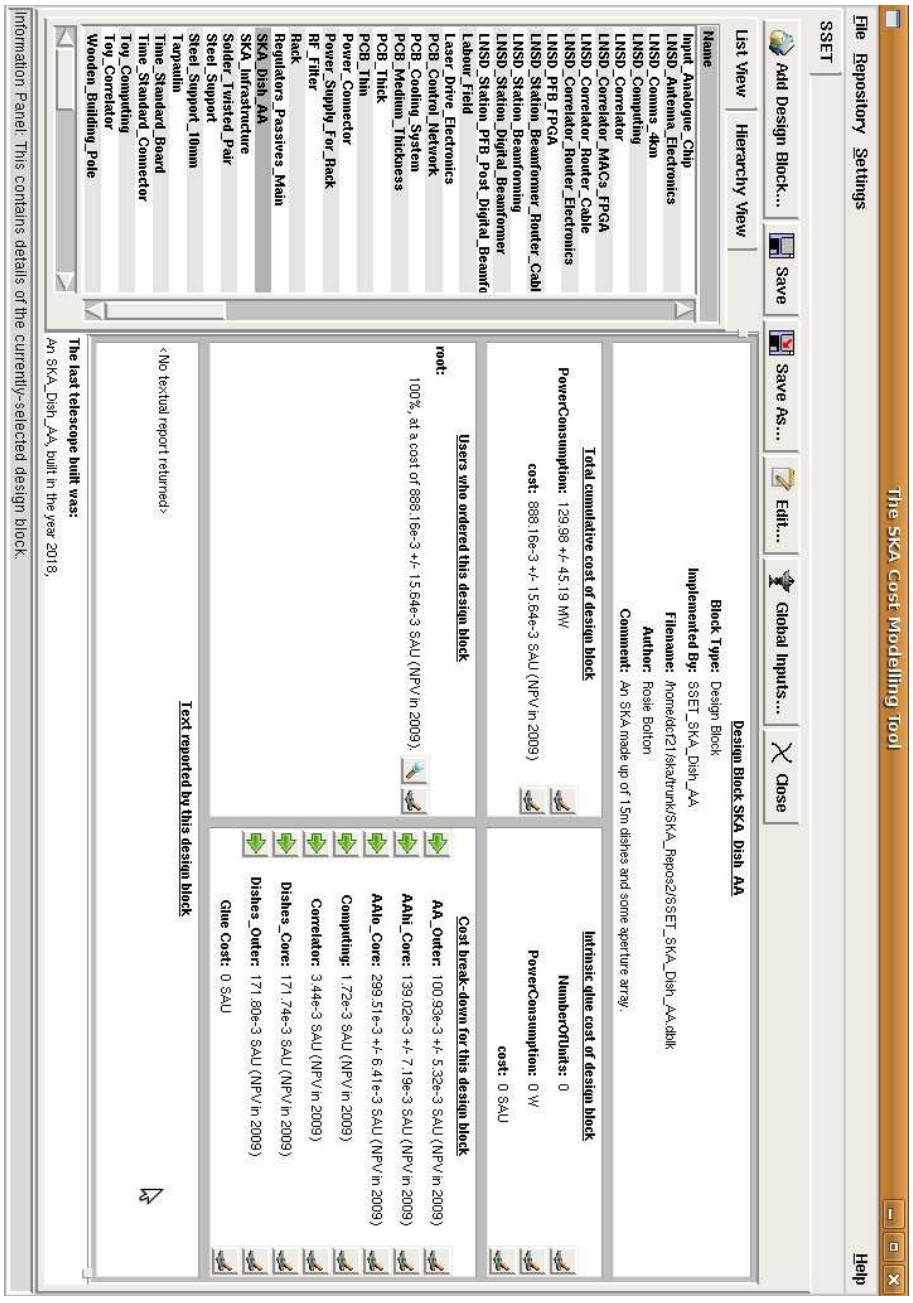


Figure 9: Summary information about the top-level SKA telescope design block. On the left, the tool lists the parent design blocks which have sent orders to it: in this case there is only one order, sent from *root*, that is, the user. On the right, the tool lists the break-down of the design block's total cost between its various children.

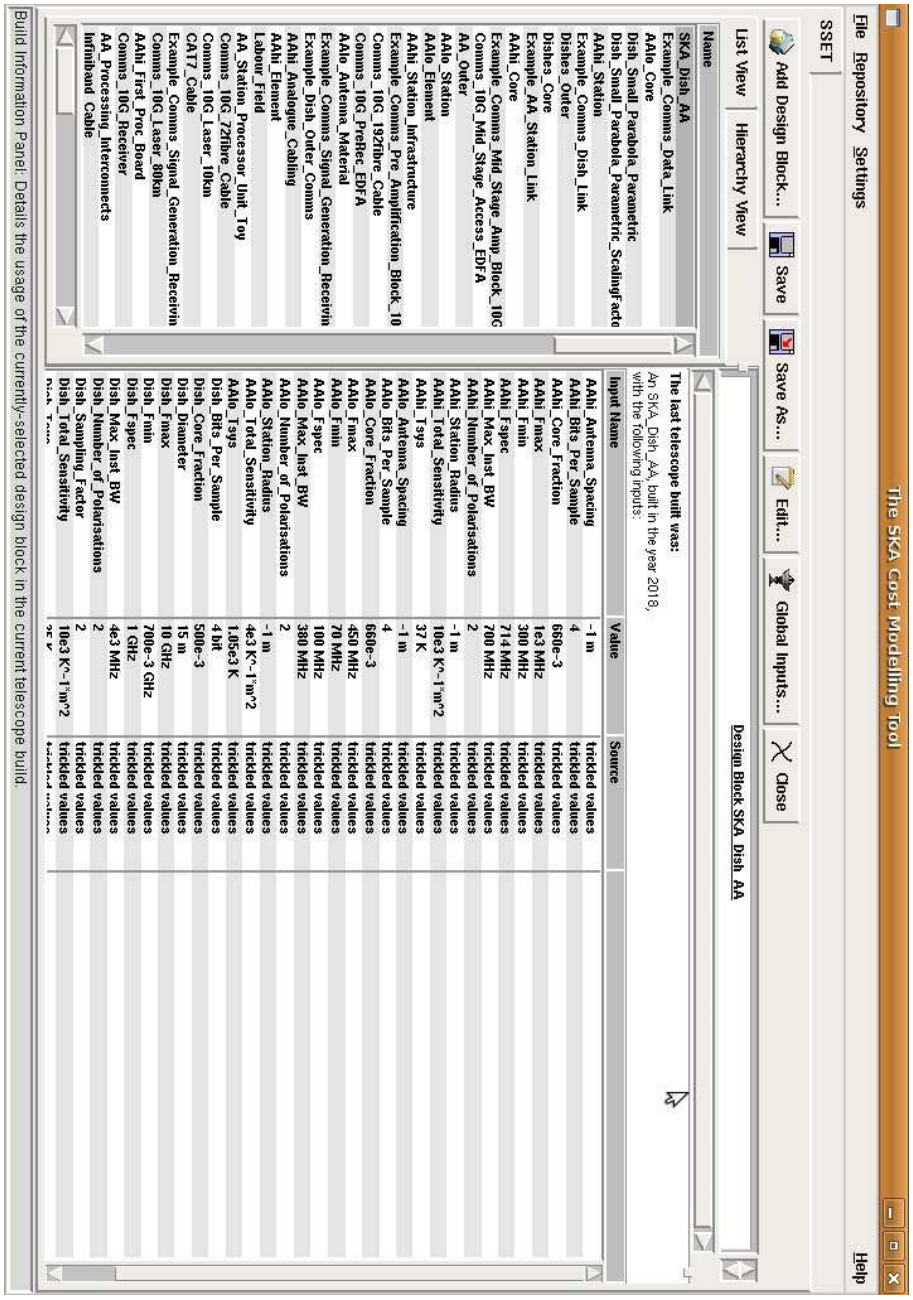


Figure 10: A spreadsheet showing the input parameters which were used in the latest telescope to be built. Also listed are the information sources from which each of the parameter values were drawn.

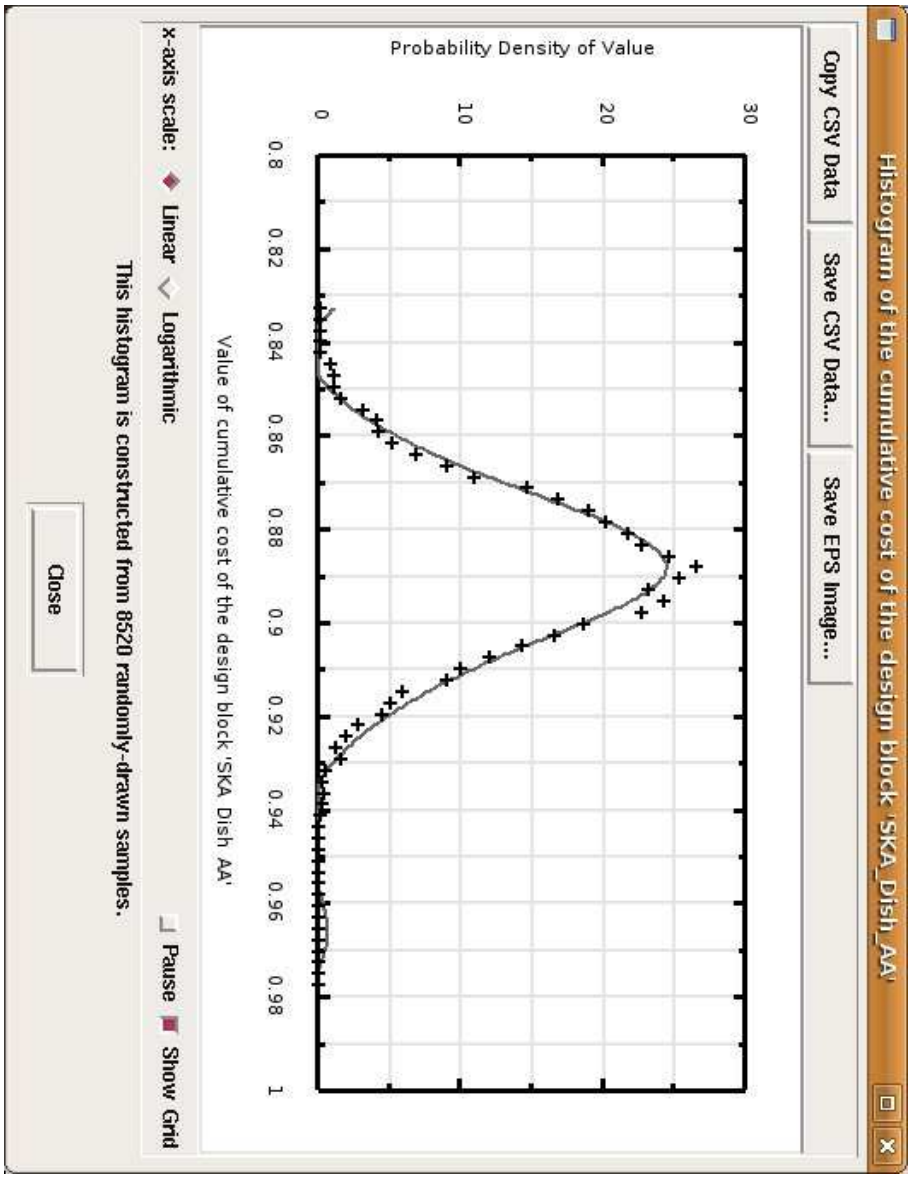


Figure 11: The tool can display the probability distribution function (PDF) for the cost of any component or design block, based upon random samples drawn from it. In this case, we show the PDF for the total cost of the SKA as measured in our ficticious currency, the SAU; the vertical axis is measured in units of probability density per unit SAU.

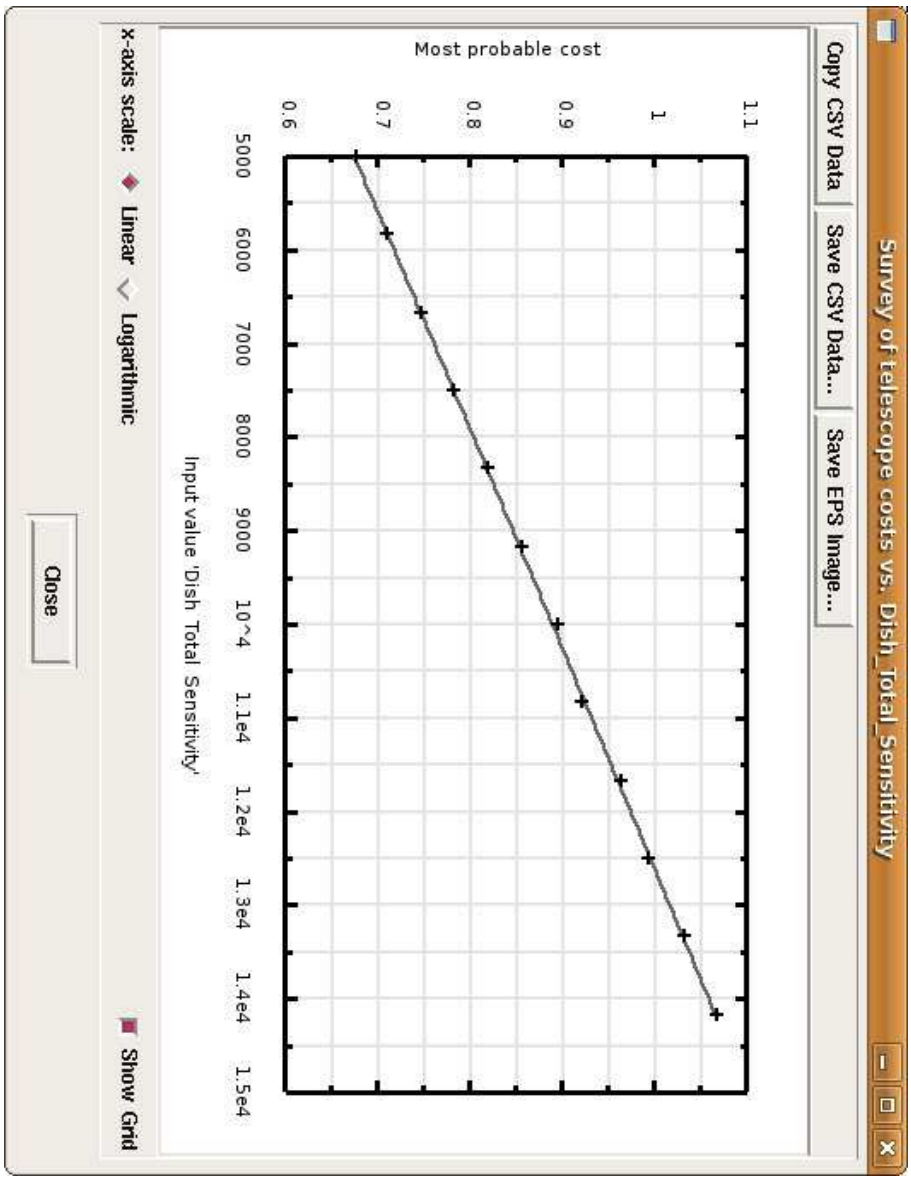


Figure 12: A parameter survey of the cost of the SKA in SAU as a function of the desired sensitivity of the dish component, measured along the horizontal axis in $m^2/K$.

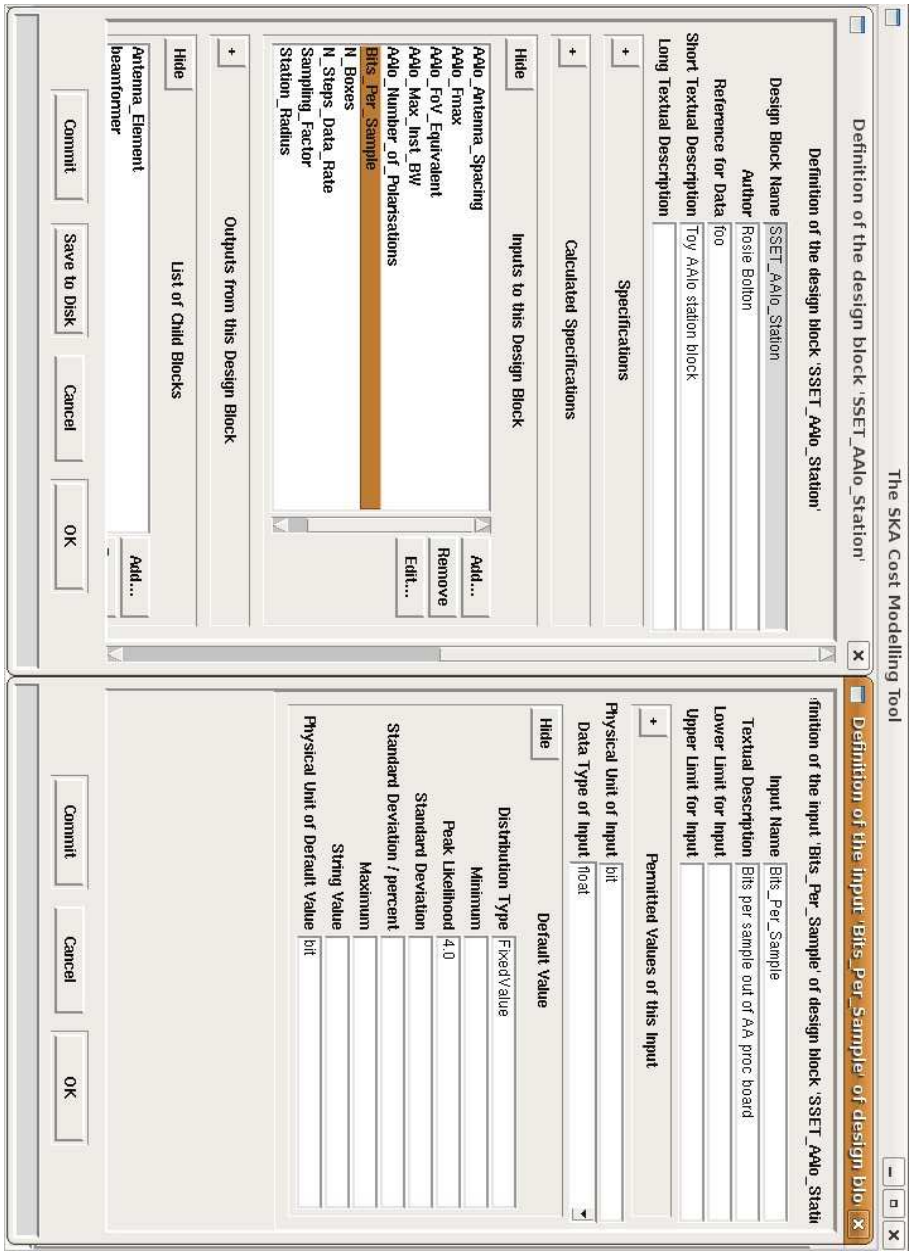


Figure 13: The Costing Tool's design block editor, here editing the *AAlo_Station* design block. The left panel shows the design block's definition, including a list of its inputs. In the right panel, we have opened the *Bits_Per_Sample* input, to show that it is measured in physical units of bits, and that its default value is 4 bits per sample.

## References

Alexander P., Faulkner A., Torchinsky S., van Ardenne A., Wilkinson P., de Vos M., Bakker L., Garrington S., Harris G., Ikin T., Jones M., Kant D., McCool R., Patel P., *SKADS Benchmark Scenario - Design and Costing*, SKA Memo 93, 2007

Bolton R., Faulkner A., Alexander P., Torchinsky S., van Ardenne A., Wilkinson P., de Vos M., Bakker L., Garrington S., Harris G., Ikin T., Jones M., Kant D., Kettle D., McCool R., Patel P., Romein J., *SKADS Benchmark Scenario Design and Costing – 2*, SKADS memo, submitted as an SKA memo, 2008

Chippendale A. P., Colegate T. M., O'Sullivan J. P., *SKAcost: a Tool for SKA Cost and Performance Estimation*, SKA Memo 92, 2007

Ford D. C., *The SKA Costing Tool Users' Guide*, , 2009

Gaensler B., *Key Science Projects for the SKA*, SKA Memo 44, 2004

Gaensler B., Lazio J., *Trade-offs Between Science and Engineering*, SKA Memo 82, 2006

Horiuchi S., Chippendale A., Hall P., *SKA system definition and costing: a first approach*, SKA Memo 57, 2004

ISPO, *Reference Design for the SKA*, SKA Memo 69, 2006

Jackson C., *SKA Science: A Parameter Space Analysis*, SKA Memo 29, 2003

Jackson C., *SKA Key Science Requirements Matrix 2006; Prime Science Drivers*, SKA Memo 83, 2006

Jones D. L., *SKA Science Requirements*, SKA Memo 45, 2004

Schilizzi R. T., Alexander P., Cordes J. M., Dewdney P. E., Ekers R. D., Faulkner A. J., Gaensler B. M., Hall P. J., Jonas J. L., Kellermann K. I., *Preliminary Specifications for the Square Kilometre Array*, SKA Memo 100, 2007

van Rossum G., *Python Tutorial*, URL `http://docs.python.org/tut`, 2008

# Index