Kenneth J. Turner and Qian Bing. Systematic Testing of Radiotherapy Accelerators. In Hartmut Ehrig, Berndt J. Kramer and Atilla Ertas, editors, Proc. Integrated Design and Process Technology (IDPT 02), pages 33.1-33.8, Society for Design and Process Science, Grandview, Texas, USA, June 2002.

Integrated Design and Process Technology, **IDPT-2002**
Printed in the United States of America, June, 2002

Systematic Testing of Radiotherapy Accelerators

Kenneth J. Turner and Qian Bing
Computing Science and Mathematics
University of Stirling, Scotland FK9 4LA
kjt@cs.stir.ac.uk, qb@cs.stir.ac.uk

ABSTRACT:
The nature of radiotherapy accelerators is briefly explained. It is argued that these complex safety-critical systems need a systematic basis for testing their software. The paper describes a novel application of protocol specification and testing methods to radiotherapy accelerators. It is explained that the accelerator control system is specified using LOTOS (Language Of Temporal Ordering Specification). It is completely infeasible to use the LOTOS directly for test generation. Instead, specification inputs are restricted using annotations in a Parameter Constraint Language. This is automatically translated into LOTOS and combined with the accelerator specification. It then becomes manageable to generate tests of the actual accelerator to check that it agrees with its specification according to the relation *ioconf* (input-output conformance). Test annotations and a sample test are described.

I. INTRODUCTION

A. Radiotherapy Accelerators

Radiotherapy equipment is used medically to deliver controlled doses of radiation to a patient, usually to destroy cancerous tissue. Among the several kinds of radiotherapy equipment, the most important is the linear accelerator ('accelerator' or 'linac'). This is so-named because it accelerates a beam of electrons to high energy that can be used directly or to generate x-rays. Accelerators are highly specialised pieces of equipment that require special housing and trained operators. For this reason, they are generally found in oncology (cancer) clinics.

Radiotherapy is a safety-critical procedure that demands accurate delivery of radiation. A number of radiation accidents have been well documented (e.g. [16], [17]). The Therac-25 accelerator is infamous as having caused accidental injuries, in some cases leading to death [18]. In fact, a radiation underdose is as undesirable as an overdose since it may fail to kill a tumour. Not delivering radiation to the exact area is also serious since it damages the surrounding healthy tissue instead of destroying the cancerous growth.

Radiotherapy equipment is regulated, designed and tested to very high standards. A review of standards for software-controlled medical devices is given in [14]. The main international standards of relevance to this paper are those in the IEC 601 series. This is a very large collection of standards, specifically including programmable electrical medical systems [8], [9]. A number of subsidiary standards concern accelerators [10]. The US Food and Drug Administration has published guidelines on Good Manufacturing Practice [4] that are relevant to software-controlled medical devices. Radiotherapy machines are typically certified in the US before they are sold anywhere in the world. The American Association of Physicists in Medicine has laid down a code of practice specifically for radiotherapy accelerators [19]. The Canadian Atomic Energy Authority also plays an active role in regulating radiotherapy devices. The European Commission is defining standards for safety of medical equipment (e.g. the Medical Devices Directive [2]). More general software development standards are also relevant, such as the ISO/IEC 9000 series on quality assurance and its European EN equivalents.

Early radiotherapy equipment was essentially hardware. Hardware aspects of accelerators are regularly and thoroughly checked. For example, dosimeters (dosage meters) are periodically calibrated against national standards. The accuracy of radiation delivery is also regularly checked in simulated treatments. The hardware is extensively protected by interlocks that address situations like power supply failure, dosimeter failure, or entry to the treatment room during radiation delivery.

Modern accelerators are, however, complex software-controlled systems. (The Therac-25 accidents stemmed in part from a reduction in the number of hardware interlocks.) Accelerator software resembles standard application software. It requires a graphical user interface, peripheral input-output, file system operations, and data communications. The accelerator software depends on a conventional style of operating system. The software must respect strict demands for dependable, real-time operation. Software, unlike hardware, does not deteriorate over time so that different reliability concerns apply. Like any application, the accelerator control software is upgraded from time to time by the manufacturers. Of course, the software is developed much more carefully than conventional application software. However with new accelerator software, it is desirable to check that the new version has not introduced any flaws. Surprisingly, there seems to be little automation to help clinics to do this.

B. Formal Methods

Formal methods are an obvious choice to support the development and testing of radiotherapy equipment. For ex-

ample formality offers precise specification, rigorous analysis, and automated test generation. Somewhat unexpectedly, radiotherapy equipment has attracted little attention from the formal methods community. [21] is one of few contributions, having made use of LOTOS (Language Of Temporal Ordering Specification [11]) to investigate equipment characteristics. The only other work known to the authors has used Z in the development of software for a radiation therapy machine [13].

Conformance testing uses experimentation to check an implementation against its formal specification. Tests are derived from the specification, then applied to the Implementation Under Test. Based on observations made during test execution, a verdict is given about the correct functioning of the implementation. The unique contribution of this paper is the application to radiotherapy accelerators of techniques normally used only with protocols: the LOTOS language, and methods for generating conformance tests.

Test theories were first studied more than a decade ago. They aim to define implementation relations by explicitly using external tests and observations (e.g. [20]). Apart from defining an implementation relation, conformance testing involves finding a set of tests to distinguish between correct and incorrect implementations. Several test generation algorithms for LOTOS-like specifications have been proposed. In [22] a testing theory is proposed for communicating systems that distinguish inputs and outputs. This is a more realistic view of systems such as accelerators.

The test suite for an accelerator is generated from a LOTOS specification following an algorithm based on that given in [22]. The authors have extended CADP (Cæsar Aldébaran Development Package) to generate accelerator test suites automatically. Each test case in the generated test suite defines the possible inputs and expected outputs.

The following sections start by giving an introduction to radiotherapy accelerators. The structure of the control system specification is explained, avoiding LOTOS details for the benefit of the general reader. The strategy for test annotation is explained with some basic examples and with the actual test constraints for an accelerator. A non-technical outline is then given of how the constrained specification is used to create a test suite automatically.

II. RADIOTHERAPY ACCELERATORS

A. Accelerator Hardware

The entire accelerator is located in a treatment room. This is heavily screened to prevent radiation leakage to the outside. Access is via an interlocked door (or gate) from the control room. The control room houses the operator console and supporting computer systems. For security, the operator must insert a key into the console before it will work. [6] provides a comprehensive introduction to the theory and practice of accelerators.

A typical accelerator is shown schematically in figure 1.

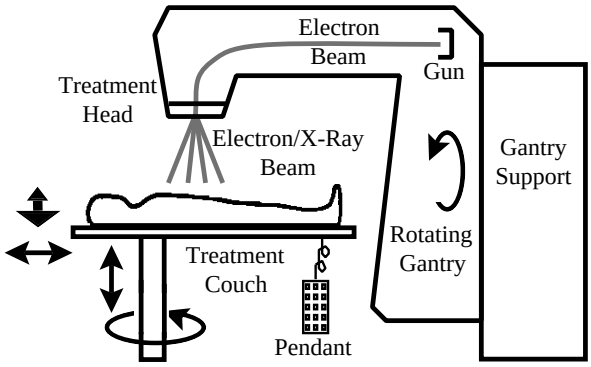

Fig. 1. Accelerator Outline

The accelerator proper is mounted on a gantry that rotates about the horizontal axis. The accelerator uses a travelling waveguide to accelerate electrons from an electron gun. The beam is controlled so as to yield electrons with energies typically in the range 6 to 20 MeV (million electron-volts). Radiation dosages are measured in MUs (monitor units). MUs reflect the calibration of dosimeters rather than any absolute unit, but 1 MU approximates to 1 cGy (centigray, a standard unit of radiation dosage).

The horizontal electron beam is bent by magnets through 90°(or 270°) so that it points downwards. In electron mode, the electrons emerge through a radio-transparent plate to reach the patient. In x-ray mode the electrons strike a target, causing a shower of x-rays towards the patient.

The treatment head contains a collimator. This consists of four movable plates, two that move in the X direction and two that move in the Y direction. They define a rectangle that restricts the beam to a defined aperture. A sophisticated accelerator will have an MLC (multi-leaf collimator). This has many (one or two hundred) individually movable leaves that may be used to set an arbitrary shape for the beam aperture. An 'accessory' may be fitted to the treatment head to control the beam distribution. The treatment head also houses an optical system that allows the shape and position of the beam to be seen on the patient's skin prior to treatment.

The patient lies on a treatment couch that may be adjusted for height, in-out position (longitude), side-to-side position (latitude), and rotation. A pendant (remote control device) is attached to the couch for setting the couch position and also for rotating the gantry. The operator sets up the patient and the accelerator so that the correct part of the body will be irradiated.

B. Accelerator Control System

During treatment, the delivered radiation dose is read periodically from the accelerator. For safety, this is measured by two independent dosimeters and each dose reading is accumulated. The first dosimeter reading is usually what determines that treatment is complete. The accumulated dose

should rise to the planned dose, but some tolerance is allowed. In case the first dosimeter does not work properly, readings from the second dosimeter are used as a backstop. Treatment is aborted if the second dose measurement exceeds the planned dose by 20 MUs or more. The dose rate is also checked at every measurement. It may not deviate from the planned rate by more than an amount that depends on the particular treatment. Finally, the treatment time is calculated from the dose and dose rate. The clock time is read to ensure that treatment does not exceed the planned time by more than 10%.

Figure 2 shows the main elements of a typical control system. The operator usually starts by arranging the patient and the accelerator geometry in the treatment room. The in-room computer displays setup information in the treatment room. The operator then retires to the control room, where treatment details are set up on the console computer. The console display shows the current accelerator setup and status. Treatments are usually planned separately and stored on a file server. The treatment is downloaded to the treatment computer and thence to the console computer.

Peripheral input-output is handled by a separate communications computer. During actual treatment (delivery of radiation), the accelerator is under the command of a separate control computer that issues commands and monitors status via common RAM. The entire system uses about half a dozen computers or microprocessors so it is not surprising that the software is complex. For the work reported in this paper, the control system has been drastically simplified as shown in figure 3. This effectively groups all of the control functions in a single black box. The main inputs and outputs are shown in the diagram.

III. Accelerator Specification

The simplified accelerator control system shown in figure 3 has been specified in LOTOS. Although the specification reflects a particular type of accelerator, the description is typical of a variety of accelerators. The specification is straightforward: 730 non-comment lines, about half of these being taken up by data types.

Many of the data types simply rename the natural numbers (e.g. dose units, angles, positions). Although in practice these parameters are floating point numbers with various scales and ranges, this simplified approach is acceptable. It just means that the offset and units for these parameters are calculated differently from clinical practice.

Although the specification contains a *Clock* process, this merely increments a time count as a natural number. This is sufficient for the purposes of the specification. It would be necessary to use E-LOTOS (Enhanced LOTOS [12]) if a more precise notion of time were required. However, E-LOTOS tool support is still rather incomplete.

The main process is *Control*. Initially this allows accelerator setup by the *Setting* process. Setting the gantry or the couch position causes movement commands to be issued, but other inputs are merely stored prior to treatment. The console display is updated after every input to reflect the current accelerator status. The operator may initiate treatment once a valid set of parameters has been entered.

The accelerator setting is then sent to the accelerator and radiation begins. The *Monitoring* process periodically reads the accelerator status, i.e. the two dosimeter readings. Normally, treatment continues until the prescribed dose has been delivered. Section II-B describes how the accumulated doses and the clock are used to terminate treatment. The operator is permitted to pause and resume treatment, perhaps because the patient is restless. Any abnormal condition such as an interlock stops the treatment immediately. On completion of treatment, radiation is stopped and the operator is informed.

IV. Test Generation

A. Test Strategy

The aim of test generation is to produce useful system tests from the accelerator specification. There does not appear to be a systematic procedure for clinics to check accelerator software. Automated testing can therefore supplement normal clinical practice, particularly following a software upgrade. Currently, the generated tests can only be run manually by an operator following a script. In future it is hoped to convert the tests into the same format as used for patient treatments. It would then become possible to load the test suite into the treatment computer and to execute the tests automatically.

There is a choice of when to generate tests. Ideally, a symbolic transition system would first be created from the specification. Tests would then be generated by traversing this transition graph, choosing test values on-the-fly. Unfortunately tools to achieve this are not yet available, though [1] is a promising basis. Tests for accelerators have therefore been generated by first constraining the specification behaviour. This is done by imposing input constraints with a special-purpose language that is automatically translated into LOTOS and composed with the specification.

Most work on formally-based (conformance) testing has concentrated on protocols. It has been shown that the same methods can be applied to hardware testing [15]. Both protocol testing and hardware testing are control-dominated. That is, the focus is on behavioural rather than data aspects. An accelerator is, however, heavily influenced by data. For example, the specification outlined in section III is controlled by fourteen input settings. Most of these have a very large number of possible values (e.g. dose, dose rate, positions, angles) that may be set in any order. As a result, the number of possible test cases is astronomical (in excess of 10^{12}). Only a small number of these test cases would be interesting.

It is therefore necessary to seriously restrict the values used for testing. Fortunately, the inputs needed by an ac-

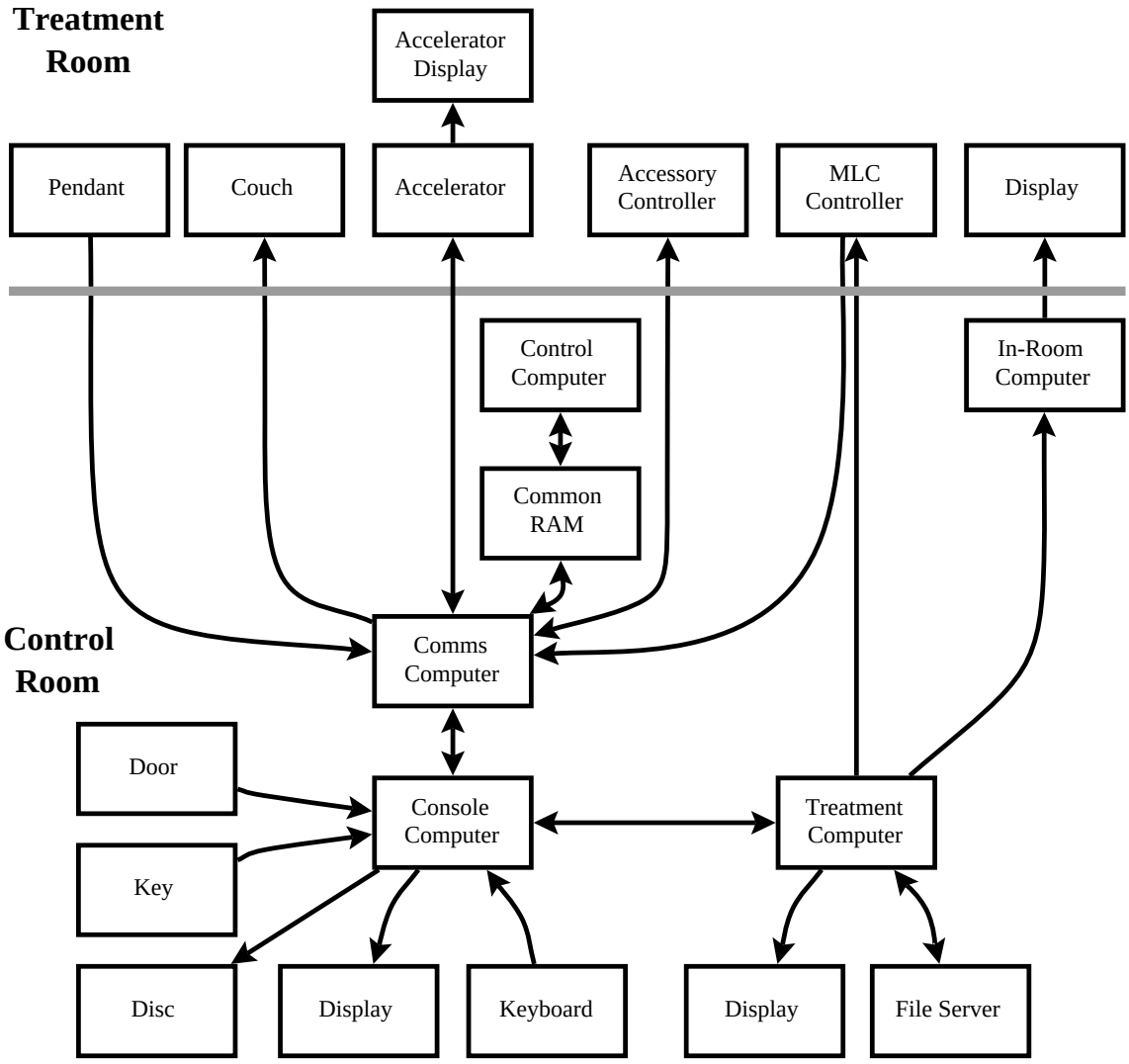

Fig. 2. Accelerator Control System

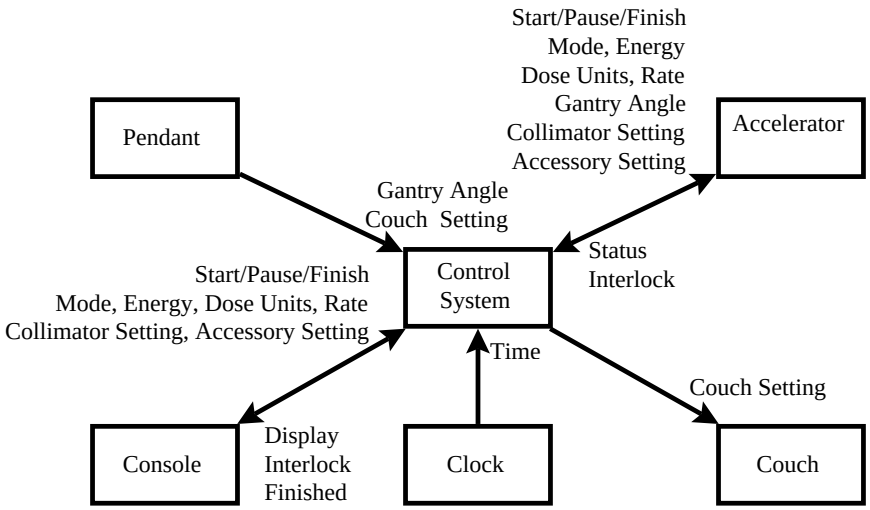

Fig. 3. Simplifi ed Accelerator Control System

celerator fall into two categories: values from a short list of alternatives, and numbers within a defined range. An input with limited alternatives (e.g. an enumerated type) can be tested in full. An appropriate technique for ranges is boundary testing as used in software engineering. As noted earlier, numeric input parameters have been mapped to natural numbers in the specification. Suppose that some input is a natural in the range 6 to 20 inclusive. Significant test values are the lowest and highest permissible values, plus some middling value (e.g. 6, 13, 20). If it is desired to check for robustness, values just outside the permissible range should also be checked for rejection (e.g. 5, 21). It is not possible to analyse an arbitrary LOTOS data type to determine that its values lie in a bounded range. Test generation therefore relies on the specifier giving some help, namely the nature and values of bounds.

Even with these restrictions to test values, the number of test cases is still far too large because of input permutations. Instead, the specifier can help by indicating the order in which inputs may be supplied. Even if this is only a partial order, the combinatorial situation can be greatly improved.

B. Input Value Constraints

Only the specifier knows the intended range and ordering for input values; these cannot (reasonably) be inferred from the specification. The authors have designed PCL (Parameter Constraint Language, 'Pickle') to allow the specifier to give guidance on what inputs to supply and when. In general, this is a difficult problem as a wide variety of event structures and specification styles are possible with LOTOS. PCL is a reasonably general and flexible language that is applicable to many other testing problems. However it was particularly inspired by the need to test the kind of accelerator specification found in section III.

PCL annotations are special LOTOS comments. As comments, they do not affect the specification behaviour. Instead, the LOTOS source is preprocessed to obtain constraints that are automatically placed in parallel with the main specification behaviour. This restricts certain inputs, while leaving outputs and some inputs unconstrained. If symbolic automata are generated in future, the PCL annotations will be used to guide the generation of test values during automaton traversal.

In fact, LOTOS does not distinguish input from output so key event occurrences must be annotated. To be exact, only one example of each event structure is annotated. Since the same data type may be used in different events with different sets of values, it is appropriate to annotate key events rather than data types. An example might be:

prime : **values**(2, 3, 5)

indicating that only the inputs 2, 3, 5 should be considered during test generation. The label *prime* for this set of values is used when input constraints are assembled. The PCL preprocessor infers the event format from the context.

The **values** directive is appropriate for a limited set of discrete values. If the parameter is a number in a range (as is very common in accelerator specifications), a shorthand is available:

hour : **range**(1, 12)

which is equivalent to the lowest, middling and highest values: 1, 6, 12. For robustness testing, the values just on each side of this range may be included by:

hour : **bounds**(1, 12)

which includes 0 and 13 as well.

There may be interdependencies among input values. Suppose that input *size* has test values 5, 13 and 21. The range for a further parameter may be given by expressions based on this:

value : **values**(*size* + 5, 2 * *size*)

Considering the test values for *size*, this equates to selecting **values**(10, 10), **values**(18, 26) and **values**(26, 42).

Input values may be given inside a LOTOS expression, in which case the expression is evaluated for each value:

status : MakeStatus(**range**(1, 10), **values**(2, 4, 8))

This is equivalent to the values *MakeStatus(1, 2)*, *MakeStatus(5, 4)* and *MakeStatus(10, 8)*.

Because the overall constraints are composed with the main behaviour, they must synchronise on every event. It is therefore necessary to annotate output events and unconstrained input events as **free** so that they can be allowed to happen. It is possible, however, to write LOTOS events whose event structure cannot be determined automatically. In such a case it is necessary for the annotation to give the generic format of the event.

C. Input Ordering Constraints

The PCL annotations so far allow the values of input events to be constrained by the specifier. To limit combinatorial explosion, it is also desirable to limit the possible orderings of inputs. Again this is done by annotating the specification. Inputs may be ordered in three different ways:

separate: input values are chosen completely independently. This is the most general case, but causes the largest number of variations to be tested.

grouped: the *i*th values for inputs occur in groups, but in any relative order. Suppose that one input may take values 2, 3, 5 while another may take values Red, Green, Blue. The first, second and third values from each are chosen and input in either order: 2 and Red, 3 and Green, 5 and Blue. Grouped inputs must have an identical number of values. This is not as restrictive as it might seem since the inputs will typically all be defined by **range** or **bounds**. Grouping significantly reduces combinatorial explosion by checking all values together.

serial: the *i*th values for inputs occur in sequence. For the same example as above, the inputs would be: 2, Red, 3, Green, 5, Blue. This is obviously the most restrictive but least expensive combination of inputs.

Each input should fall into one of these three categories (or conceivably be **free** if it is unconstrained). To limit combinatorial explosion further, **separate** or **grouped** values may be included in a **serial** list. As a further enhancement, an input may be followed by a question mark to indicate an optional value. Consider a hypothetical stock control system with inputs annotated as follows:

separate(price, tax?);
grouped(colour?, size);
serial(weight, **separate**(type, code?), stock, postage?)

meaning that inputs *price* and optional *tax* may be chosen independently. Input values for optional *colour* and *size* should be chosen in groups. An input value for *weight* is followed in either order by *type* and optional *code*, then inputs are provided for *stock* and optional *postage*.

Annotations for input ordering are given following the top-level LOTOS behaviour expression. The annotations are extracted by the preprocessor after all the input value annotations and used to automatically generate an overall constraint process in LOTOS. This results in a new specification that is used to generate tests. Depending on the input combinations, a number of LOTOS behaviour patterns are required for the constraint processes.

A final complication is that the specification may have cyclic behaviour, so it is necessary to know when a fresh set of input values should be generated. It is assumed that some key event can be annotated with **finish** as marking the end of the current cycle.

D. Accelerator Specification Annotations

The following input value annotations were placed in strategic places in the accelerator specification. They are scattered but are extracted by the preprocessor:

```
mode : values(                              (* treatment *)
  XRayMode, ElectronMode)
energy : range(6, 20)                       (* beam energy *)
dose : range(5, 100)                        (* dose units *)
rate : range(1, 50)                         (* dose rate *)
gantry : range(0, 359)                      (* gantry angle *)
x1 : values(0, 0, 39)                       (* collimator X1 *)
x2 : values(1, 40, 40)                      (* collimator X2 *)
y1 : values(0, 0, 39)                       (* collimator Y1 *)
y2 : values(1, 40, 40)                      (* collimator Y2 *)
accessory : values(                         (* accessory *)
  AccessoryIn, AccessoryOut)
rotation : range(0, 359)                    (* couch rotation *)
latitude : range(0, 50)       (* couch latitude position *)
longitude : range(0, 150)(* couch longitude position *)
vertical : range(60, 170)   (* couch vertical position *)
accelerator : values(         (* dosimeter pair readings *)
    MakeStatus(values(2, 1, 2), values(2, 1, 2)),
    MakeStatus(values(0, 25, 28), values(10, 26, 35)),
    MakeStatus(values(0, 1, 3), values(10, 50, 70)))
```

If software is governed by a bounded parameter, it is common experience that coding errors are most likely at the extremes of the range. This is the basis of boundary value testing. The same principle applies to accelerator software testing. In addition, the extreme values also check that the hardware is correctly operating over its full range. Most accelerator parameters are therefore defined by range tests.

Nearly all the test constraints above are straightforward. The dosimeter readings are given as three set of values to be used on each successive treatment. Each set of values gives three pairs of dosimeter readings. The values are chosen so that treatment stops on the final value of each triple.

Some inputs and all outputs are marked as unconstrained. In a few cases it is necessary to indicate the event structure explicitly as it cannot be determined from the context. Internal events (clock signals) do not need to annotated as they are not externally visible.

Finally, the permissible combinations of inputs are given:

```
serial(                                     (* sequence of inputs *)
  separate(mode?, accessory?),              (* either order *)
  energy, dose, rate,                       (* energy, dose, rate *)
  gantry?,                                  (* gantry *)
  x1, x2, y1, y2,                           (* collimator *)
  rotation?, latitude?, longitude?, vertical?,  (* couch *)
  accelerator)                              (* status *)
```

The names here refer to the input value labels given earlier. It will be noted that a number of inputs are optional because they have default values that need not be set.

The fairly compact annotations for input values and ordering are turned automatically into some complex constraint processes. An overall constraint process is generated automatically and placed in parallel with the main behaviour. The gates of this top-level process are inferred from the structure of the annotated events. The *Constraints* process relies on the automatically generated sub-processes *ConstraintsFree* and *ConstraintsRepeated*.

ConstraintsFree permits free events to occur without restriction. *ConstraintsRepeated* allows grouped inputs to occur independently of separate and serial inputs. If ranges are specified for the constrained events, the lowest, middling and highest values are chosen in sequence. Since the specification must be tested for various cycles of behaviour, the console *Finished* event is used as the trigger to choose a new set of repeated events.

The *ConstraintsRepeated* process uses automatically generated sub-processes. Grouped inputs are defined by *ConstraintsGrouped*, separate inputs by *ConstraintsSeparate*, and serial inputs by *ConstraintsSerial*.

E. Accelerator Test Generation

By using PCL annotations, the accelerator behaviour under test is restricted to a manageable extent. Standard test generation techniques can then be applied.

Specification languages often model a system as an LTS (Labelled Transition System). In real-world systems, however, inputs and outputs are clearly distinguished. The inputs of a system are always enabled and cannot refuse the

actions offered by the environment. After the system consumes an input and produces its outputs, the environment has to accept the outputs. Communication is thus no longer symmetric. Following [22], this kind of behaviour is modelled as an IOLTS (Input-Output Labelled Transition System).

Several implementation relations have been defined to express conformance of an implementation to its specification. In these relations a specification is modelled as an LTS, and an implementation as an IOLTS. This is because an LTS can give a more abstract view of a system, while an IOLTS is closer to reality. The relation *ioconf* (input-output conformance) is appropriate for accelerator specifications. This relation judges an implementation to be correct if, after every trace of the specification, the implementation outputs can also be produced by the specification. An implementation cannot produce outputs that are not expected by the specification. Since this also holds in quiescent states, the implementation may not output if the specification cannot do so. (A quiescent state is one in which no output is expected.)

Checking *ioconf* can be achieved by checking trace inclusion on the suspension automaton generated from the LTS. Briefly, a suspension automaton is a directed graph built by determinising the LTS and marking quiescent states. The algorithm to transform an LTS into a suspension automaton has been described elsewhere [22] and is not repeated here.

Test generation is achieved by traversing the suspension automaton. The application of PCL annotations to the accelerator specification ensures that the behaviour is finite. Test generation aims to cover all transitions in the automaton. Generating a sequence that visits every edge in the graph at least once is the Chinese postman problem [3]. As suspension automata may not be strongly connected, the approach of [7] is adopted because it is suitable for all kinds of directed graph. This method uses depth-first search whenever possible. But when an unvisited edge cannot be reached, then breadth-first search is used to find a state with an unvisited edge. The whole procedure repeats until all transitions have been covered.

Each transition tour is a test case and is saved in a test file. The test generation algorithm may find that a state offers alternative outputs with the same gate but different values. These outputs are marked when the corresponding test cases are generated, meaning they are not necessarily matched by the implementation. Execution of the test suite takes this implementation freedom into account.

CADP (Cæsar Aldébaran Development Package [5]) supports an application programming interface that allows user-written programs to manipulate the state space of a given LOTOS specification. As reported in [15], the *TestGen* tool has been developed to generate a test suite by creating and traversing a suspension automaton. This tool was originally developed to generate hardware tests, but the same approach is applicable to generating accelerator tests.

The accelerator specification in section III was constrained as described in section IV-D. The resulting LTS, minimised with respect to observational equivalence, has 8616 states and 11300 transitions. There are 67 distinct paths and thus test cases, the longest consisting of 136 events though most are much shorter. A typical test case provides the following inputs:

mode: electron mode
energy: 13 MeV (million electron-volts)
dose: 52 MUs (monitor units)
rate: 25 MUs/minute
collimator: X1 0 cm, X2 40 cm, Y1 0 cm, Y2 40 cm
gantry: 180°
couch: rotation 180°, latitude 25 cm, longitude 75 cm, vertical 115 cm
dosimeters: 0 and 10 MUs, 25 and 26 MUs, 28 and 35 MUs

All inputs but the last are provided during treatment setup. The last gives the pairs of dosimeter readings during treatment. These are aggregated to give *cumulative* dose readings of 0 and 10 MUs, 25 and 36 MUs, 53 and 71 MUs. In the test above, treatment stops when the accumulated dose becomes 53 MUs (slightly beyond the planned figure of 52 MUs). The test would expect treatment to be aborted if the second dosimeter exceeded the tolerance level of 72 MUs. If the treatment time had exceeded the planned time, the test would also expect treatment to be aborted. Control inputs and status outputs (not shown above) are also included in the test suite according to the accelerator specification.

V. CONCLUSION

Radiotherapy accelerators have been briefly described. Since these are complex, software-controlled, safety-critical systems it is highly desirable to employ systematic tests of the control system. The structure of a typical accelerator specification in LOTOS has been outlined.

To have any hope of generating realistic tests, it is necessary for the specifier to annotate the specification with guidance on choosing useful test inputs. PCL annotations define key test inputs – either explicit values (say, for an enumerated type) or boundary test values (for a numeric range). Unconstrained events are also marked. Further PCL annotations define how inputs should be combined to make test generation practicable. The resulting constraint process in parallel with the main behaviour allows a manageable automaton to be generated. A suspension automaton is generated from this and traversed to create test cases that form the accelerator test suite.

Although PCL has been designed to help with accelerator testing, it is of general utility and should be useful for other domains. For example, LOTOS has been used in fields such as telecommunications, hardware design, graphics, and general software design. In all of these, it would be convenient to restrict test values during test generation.

So far, test generation has been based on the simplified accelerator model in figure 3. Future work will generate tests from more detailed accelerator models such as in figure 2. The major challenge will be to deal with the much larger state space. However the finer level of modelling will hopefully allow more subtle tests to be generated. The possibility of executing test suites automatically will also be investigated. Although this work is in the future, it is hoped that the paper has given sufficient insight into the practicability and importance of the approach for testing radiotherapy accelerators.

ACKNOWLEDGEMENTS

This paper reports results from the CONFORMED project (Conformance of Radiological/Medical Devices, www.cs.stir.ac.uk/∼kjt/research/conformed.html). This is being undertaken by the authors with financial support from NCC (National Computing Centre, Manchester UK, *www.ncc.co.uk*). Dr. Hamish Porter (Western General Hospital, Edinburgh UK) was generous with advice on accelerator design and operation. However any errors and misconceptions in the paper are due to the authors. Dr. Jan Tretmans (University of Twente) and Dr. Ji He are also thanked for their insights into test generation.

REFERENCES

[1] M. Calder and C. E. Shankland. A symbolic semantics and bisimulation for full LOTOS. In M. Kim, B. Chin, S. Kang, and D. Lee, editors, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE XIV)*, pages 184–200. Kluwer Academic Publishers, London, UK, Sept. 2001.

[2] EC. Medical devices directive. Technical Report 93/42/EEC, European Commission, Brussels, Belgium, June 1993.

[3] J. Edmonds and E. L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1972.

[4] FDA. Medical devices: Current good manufacturing practice. Technical Report 61 FD 195, US Food and Drug Administration, New York, USA, Oct. 1996.

[5] J.-C. Fern´andez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (CÆSAR/ALDÉBARAN Development Package): A protocol validation and verifi cation toolbox. In R. Alur and T. A. Henzinger, editors, *Proc. 8th. Conference on Computer-Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 437–440. Springer-Verlag, Berlin, Germany, Aug. 1996.

[6] D. Greene and P. C. Williams. *Linear Accelerators for Radiation Therapy*. IOP Publishing Ltd., Bristol and Philadelphia, 1997.

[7] R. C. Ho, C. H. Yang, M. A. Horowitz, and D. L. Dill. Architecture validation for processors. In *Proc. 22nd. Annual International Symposium on Computer Architecture*, 1995.

[8] IEC. *Medical Electrical Equipment – Part 1: General Requirements for Safety*. IEC 601-1. International Electrotechnical Commission, Geneva, Switzerland, 1988.

[9] IEC. *Medical Electrical Equipment – Part 1: General Requirements for Safety – 4. Collateral Standard: Programmable Electrical Medical Systems*. IEC 601-1-4. International Electrotechnical Commission, Geneva, Switzerland, 1988.

[10] IEC. *Medical Electrical Equipment – Part 2: Particular Requirements for Safety*. IEC 601-2. International Electrotechnical Commission, Geneva, Switzerland, 1988.

[11] ISO/IEC. *Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*. ISO/IEC 8807. International Organization for Standardization, Geneva, Switzerland, 1989.

[12] ISO/IEC. *Information Processing Systems – Open Systems Interconnection – Enhanced LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*. ISO/IEC 15437. International Organization for Standardization, Geneva, Switzerland, Apr. 2000.

[13] J. Jacky, J. Unger, M. Patrick, D. Reid, and R. Risler. Experience with Z developing a control program for a radiation therapy machine. In J. P. Bowen, editor, *Proc. 10th. International Conference of Z Users*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, Dec. 1996.

[14] J. Jacobson and O. Andersen. Software controlled medical devices. Technical Report SP-Rapport 1997:11, European Network of Clubs for Reliability and Safety of Software, Apr. 1997. ISBN 91-7848-669-6.

[15] Ji He and K. J. Turner. Protocol-inspired hardware testing. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *Proc. Testing Communicating Systems XII*, pages 131–147, London, UK, Sept. 1999. Kluwer Academic Publishers.

[16] E. J. Joyce. Accelerator linked to fi fth radiation overdose. *American Medical News*, 1, Feb. 1987.

[17] C. J. Karzmark. Procedural and operator error aspects of radiation accidents in radiotherapy. *International Journal of Radiation Oncology Biological Physics*, 13:1599–1602, Jan. 1987.

[18] N. Leveson and C. S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7):18–41, July 1993.

[19] R. Nath, P. J. Biggs, F. J. Bova, C. C. Ling, J. A. Purdy, J. van de Geijn, and M. S. Weinhous. AAPM code of practice for radiotherapy accelerators. *Medical Physics*, 21(7):1093–1121, July 1994.

[20] R. D. Nicola. External equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[21] M. H. Thomas. The story of the Therac-25 in LOTOS. *High Integrity Systems Journal*, 1(1):3–15, Feb. 1994.

[22] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software Concepts and Tools*, 17:103–120, 1996.