

The statechart design of a novel date input mechanism

Fintan Culwin
Faculty of Business, Information management & Computing
London South Bank University
London SE1 0AA

fintan @ lsbu.ac.uk

Keywords

Statechart, date input, empirical usability, web usability

Abstract

The statechart design of a novel date input mechanism is presented and its advantages over existing mechanisms commented upon. The use of the design, and implementation, as an introductory example of using statecharts is proposed.

Introduction

Statechart notation, as originally proposed by (Harel 1988), is now an integral part of UML (Rumbaugh et al 1999). The notation is generally used in the context of real time systems (Köhler et al 2000, Grigg & Henderson 2000), although some authors including Culwin (1994, 1998, 2000) Horrocks (1998) and Samek (2002) have demonstrated its utility in the field of interaction design, including real world examples such as traffic lights, telephones and compact disc players. Examples of its use in interaction design generally concentrate upon highly interactive artefacts which have a graphical user interface. This paper provides a short case study of the use of the notation to design, and inform the implementation of, a novel and engaging text based date input mechanism. It is suggested that by removing the additional complexities of real time systems and graphical user interface considerations the case study will be more suitable for introducing the notation.

Date Input Mechanisms

The requirement to input a calendar date is common to a wide variety of tasks and with the prevalence of Web hosted forms requires walk up usability capability. Despite the commercial importance of this requirement there have been very few published studies regarding the efficacy and efficiency of different mechanisms (Bainbridge 2002) (Culwin & Faulkner in prep). Those that have been published, although based upon very small samples, suggest that there is a high error rate.

A survey of different mechanisms contained in (Culwin & Faulkner in prep) reported that text input mechanisms were the most prevalent, closely followed by drop down menu mechanisms; with calendar mechanisms hardly represented. Examples of these mechanisms are illustrated in Figure 1. This study suggested that in most contexts the calendar mechanism would be most effective, in preventing errors, although not the most efficient, in the time and number of actions required to enter a date. Despite this Web page designers seem reluctant to use this

mechanism possibly regarding the amount of space that it occupies as excessive and so favouring the pull-down and text input mechanisms.

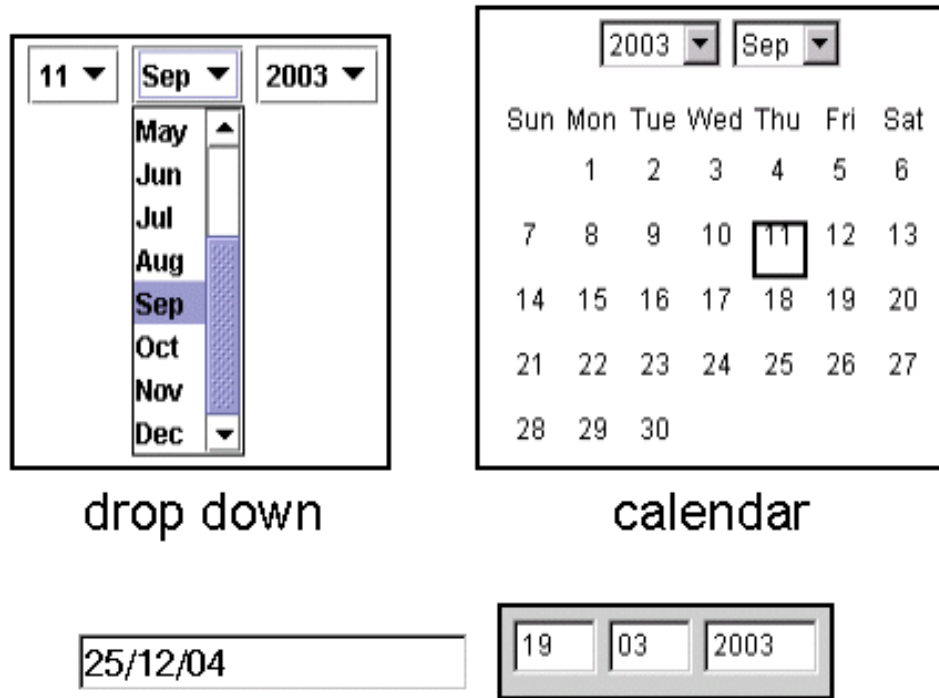
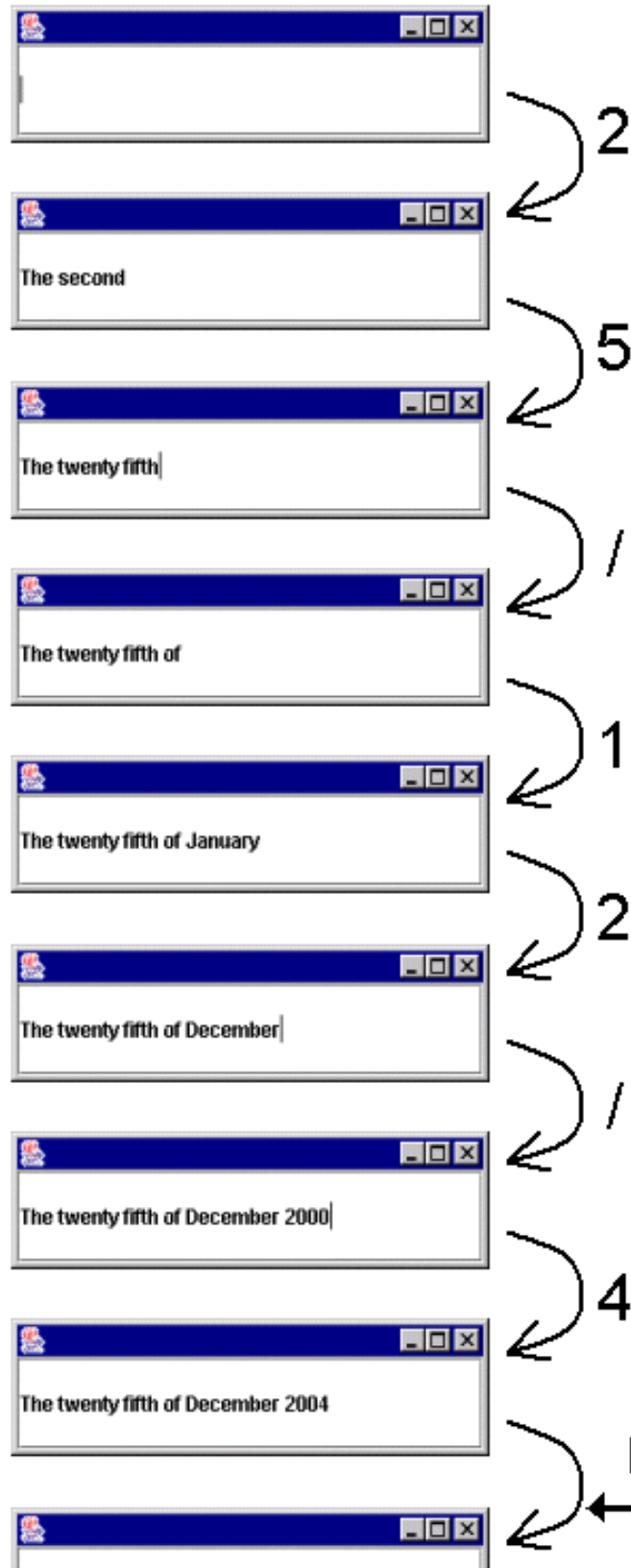


Figure 1 Date input mechanisms

There is one circumstance where text input is clearly preferable to a calendar or pull-down mechanism. For blind or partially sighted users a text input mechanism can interact much more effectively with a screen reader. However the inherent difficulties of using a text input mechanism remain. These difficulties include the inherent confusion regarding the three global date formats: dd/mm/(yy)yy, as in the UK, mm/dd/(yy)yy, as in the US, and yyyy/mm/dd, as in Japan. This confusion can be partly ameliorated by using three text input fields. It can be further ameliorated by requiring an abbreviated month name, as in 'Dec', to be used; but this then raises further problems with internationalisation. A further problem is that simple text input mechanisms do not usually deploy sufficient intelligence to detect impossible dates such as 31/10/04 (i.e. the thirty first of September) or 29/02/05 (i.e. the twenty ninth of February 2005).

The user's task context usually defines dates as 'the fourth Thursday in December' or 'two weeks on Thursday or 'the 25th of December'; rather than 25/12/04 (or 12/25/04 or 04/12/25). The common contexts are more readily served calendar rather than text or pull down date input mechanisms. These considerations, and the results of empirical investigations, led to the suggestion that the optimal input mechanism for most circumstances would be a text input linked to a calendar (Culwin & Faulkner in prep). This would allow the use of whichever mechanism was more favoured by a particular individual, accommodating naturally to whatever task context they might have.

The survey study also produced proposals for novel mechanisms one of which, known as intelligent text input, was suggested as an alternative to existing non-, or limited-, intelligent text input mechanisms. It was hoped that the mechanism would reduce the number of errors made by giving a more natural representation of the date being entered, whilst ensuring efficiency by minimising the number of keypresses made. Figure 2 illustrates the operation of the mechanism during the entry of the date 25/12/2004, also known as 'Saturday the twenty fifth of December 2004'.



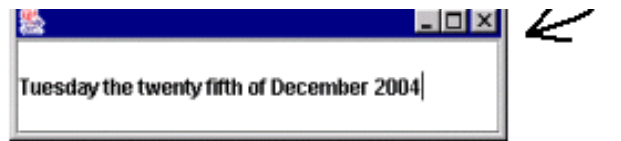


Figure 2 Intelligent Date Entry in Operation

The user has entered the sequence of keypresses '25/12/4' followed by <enter> and the system has attempted to anticipate what they might be intending. For example if the intention of the initial '2' was the designate the second of the month it could have been followed by the '/' delimiter. Likewise if the '1' following the delimiter was intended to designate the first month, January, it could also be followed by the delimiter. Hence the date 'Thursday the second of January 2004' could have been entered using the sequence '1/2/4<enter>'.

The design of the intelligent text input mechanism

Figure 3 presents the statechart for the entry of the day of the month part of a date. From the *initial*, blank, state any of the digits from '1' to '9' will initiate a transition. There are three possible paths, pressing '1' or '2' will lead to the *first or second* state where the component would display 'The first' or 'The second'. From this state pressing any of the digits from '0' to '9' would lead to the *ten or twenty something* state, where the component would display the appropriate textual representation. The path initiated by pressing the '3' key leading to the *third* state differs from the previous two states, in that its onward transition only allows a '0' or '1' key press initiating a transition to the *thirty something* state, showing 'The thirtieth' or 'The thirty first'. From the *initial* state the final pathway is followed when any of the digits from '4' to '9' is pressed, leading to the *fourth to ninth* state. As each of these keypresses alone unambiguously represents a day of the month there is no onward transition directly from this state.

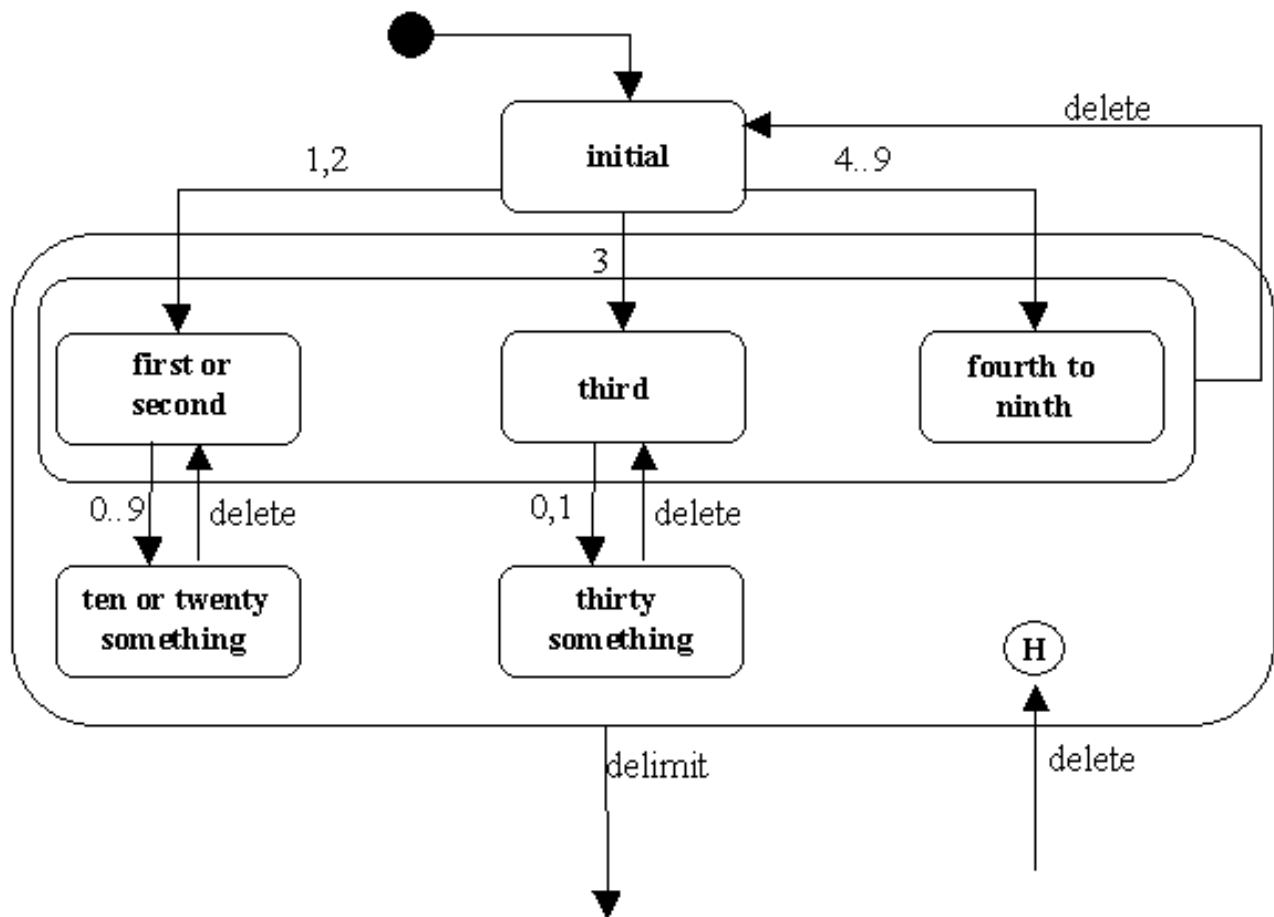


Figure 3 Intelligent date entry statechart, day of the month entry part

From any of the five non-*initial* states the delete key can be used to back up one state. For the three states directly connected to the initial state this is shown as a transition from a bounded state. In the pedagogic exposition of the design this could be shown as a simplifying refinement of a version that included an explicit transition from each of the three states.

Likewise from any of the bounded five non-*initial* states the delimit key (‘/’) can be used to indicate completion of the entry of the day of the month part of the date. This transition leads to the *initial month* state which will be fully described below. This state is visually distinguished from the five states that can lead to it by the addition of the phrase ‘of’ to whatever is shown in the previous state, as shown in the fourth part of Figure 2. An explicit design decision was taken not to have an automatic transition from each of the three unambiguous day of the month states (*ten or twenty something*, *thirty something* and *fourth to ninth*) to the *initial month* state. The rationale for this decision was to ensure consistency, having a delimiter needed for all days of the month rather than only some. Consistency would also require an explicit delimiter to move from entering the day of the month to entering the month and from entering the month to entering the year.

This statechart also illustrates a history transition returning into this collection of states. This transition originates from several of the month entry states and is followed when the user is indicating that they wish to return from entering a month to correct the entry of a day. As such the mechanism should return to the same state that it was in before the delimiter was pressed, a consideration which is expressed in a history transition as shown.

One final pedagogic observation that might be made concerning the exposition of this as an initial statechart is that it allows the differences between a visual and a logical state to be expounded. A state in a statechart is defined as 'a place in the behavioural space of an artefact which is visually **and** behaviourally distinct' (Culwin 1994). A very naïve approach to constructing a statechart for this requirement might be to attempt to encompass thirty one states, one for each possible day of the month. This chart clearly illustrates that a state can encompass a number of visually distinct but behaviourally identical situations.

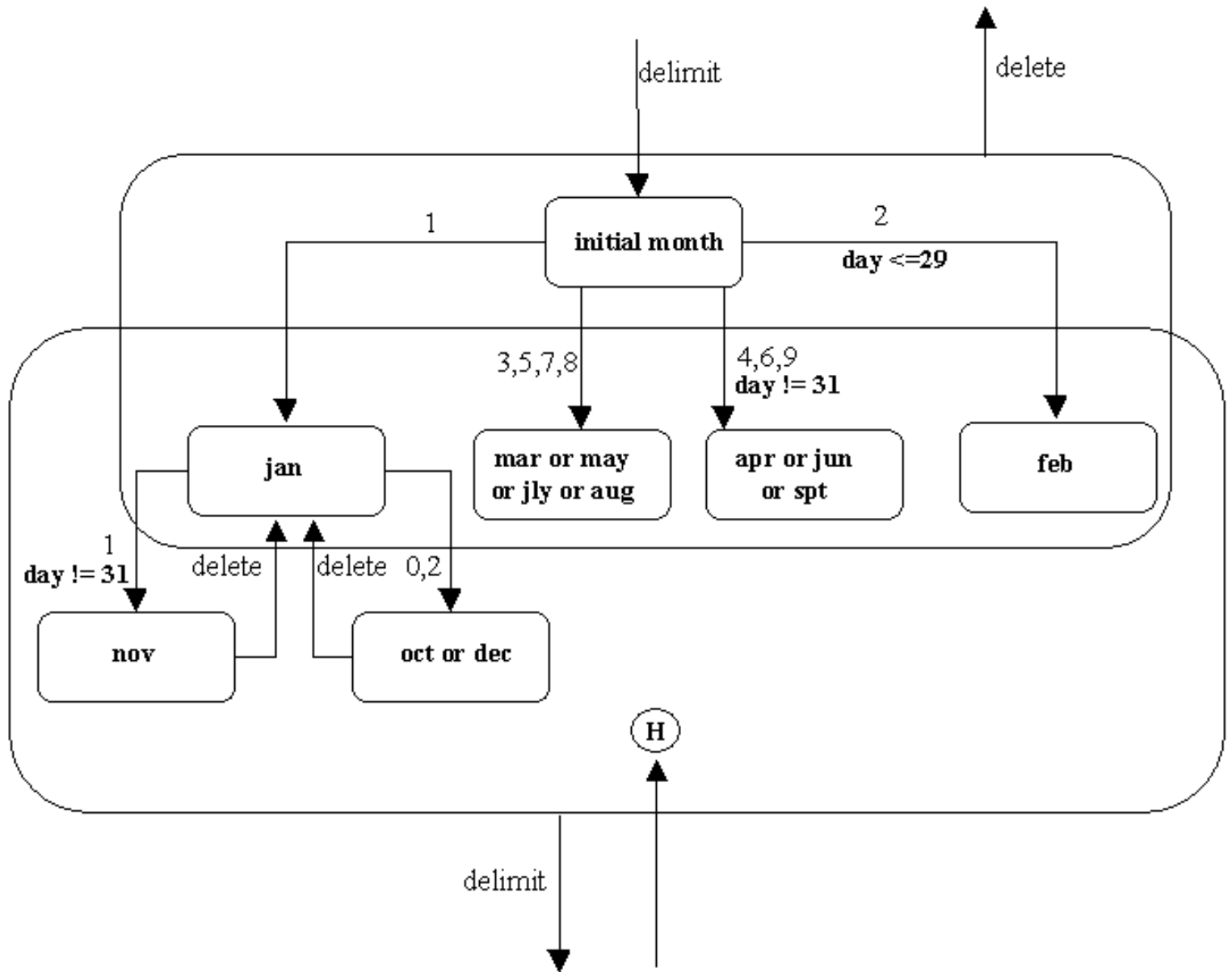
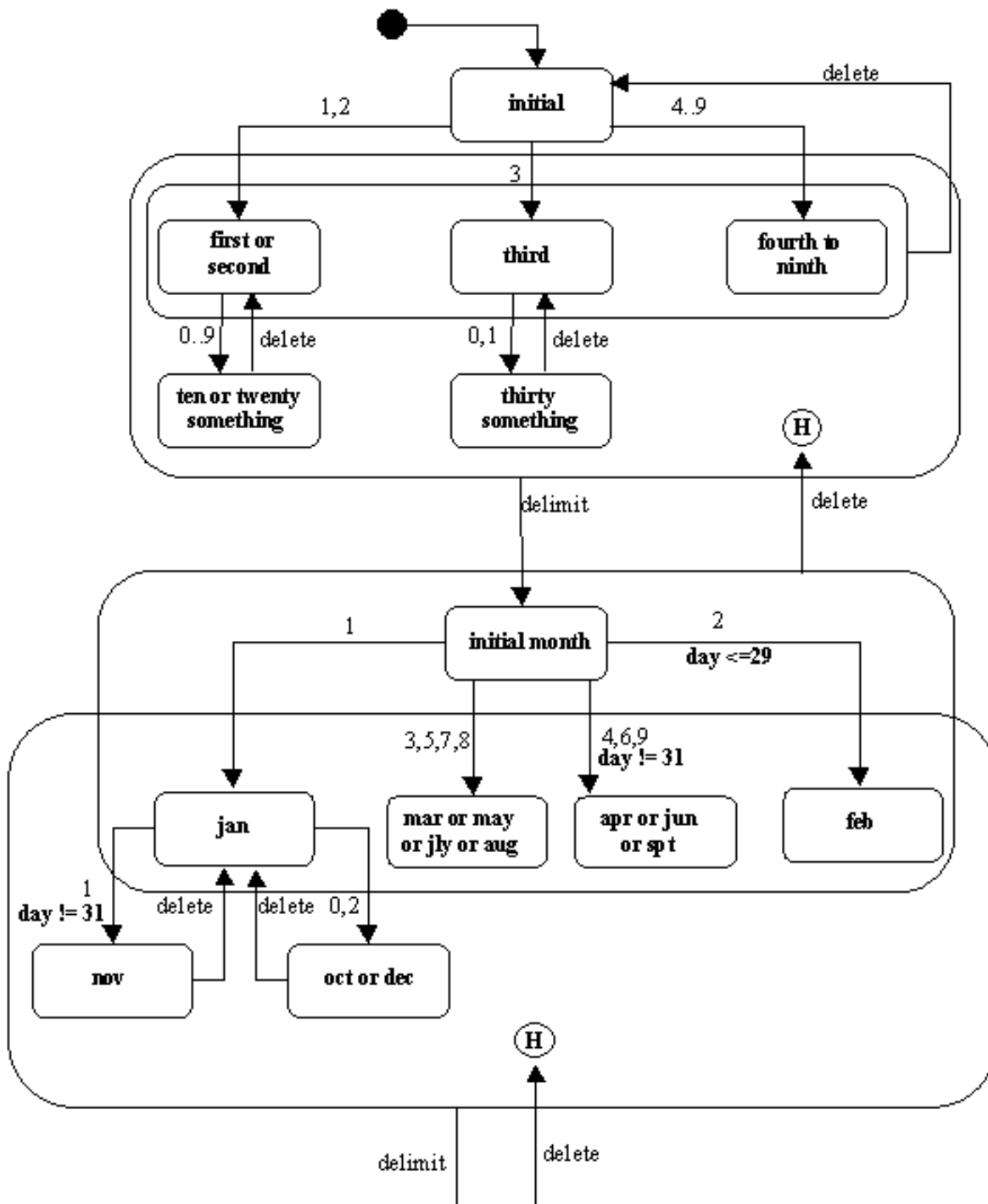


Figure 4 Intelligent date entry statechart, month entry part

Figure 4 presents the statechart for the entry of the month. The major conceptual refinement introduced in this chart is the use of a conditional on transitions. Hence the transition from the *initial month* state to the *feb* state

occasioned by pressing the '2' digit will only be followed if the value of the day of the month, as already entered, is less than or equal to 29. Likewise the transition occasioned by digits '4', '6' and '9' from the *initial month* state to the *apr or june or sept* state, and the transition from the *jan* state to the *nov* state occasioned by the '1' digit; will only be followed if the day of the month is not 31. The implements the part of the childhood rhyme '*thirty days have September, \ April June and November.*' preventing the mechanism from entering a situation where is was showing an explicitly invalid day of the month and month combination.

As the user interacts with the mechanism the appropriate name of a month is appended to what is already shown when it is in the *initial month* state. The broad design of this part of the mechanism is informed from the comparable parts of the design of the day of the month part. That is the two stage entry of those months that require two digits, a bounded transition to the year entry part of the behaviour when the delimiter key is pressed in an appropriate state and a history transition returning to this state from the year entry part.



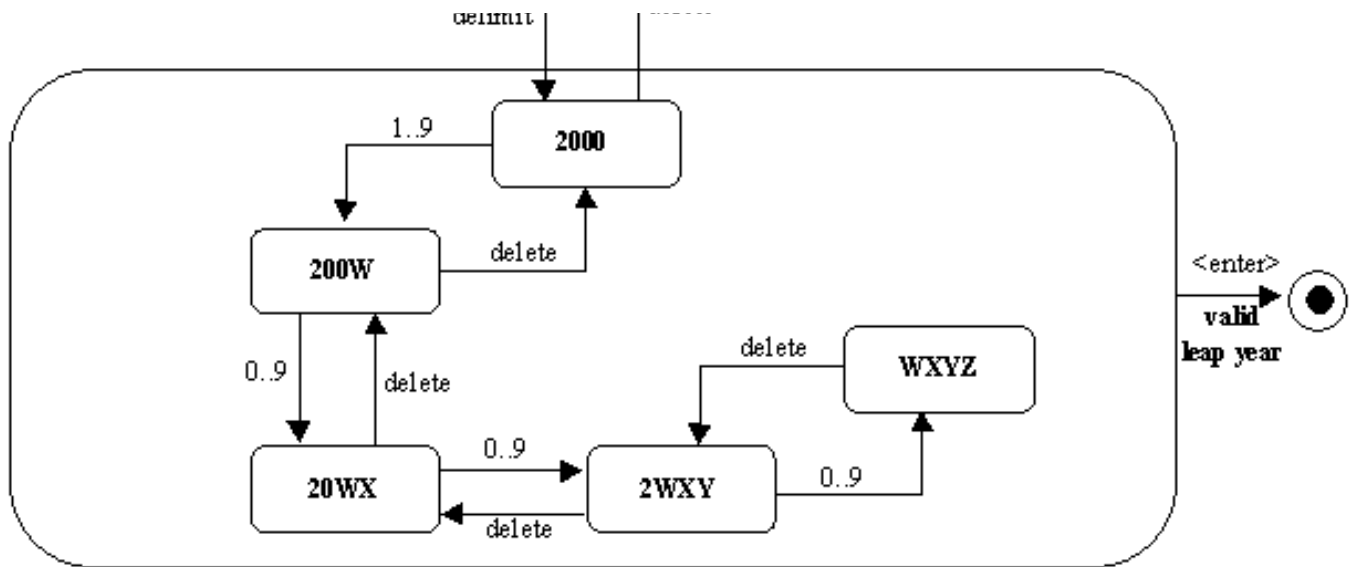


Figure 5 Intelligent date entry statechart, complete

Figure 5 presents the statechart for the year entry part of the behaviour and, also for convenience, the day of the month and month entry parts, so as to provide the complete design. The essence of the design of this part of the mechanism differs from that of the other parts partly because the nature of the task is different, having to do with a four digit year rather than a one or two digit day or month. But also in order to optimise the user's interaction with regard to the wider context of its use. The survey paper (Culwin & Faulkner in prep) revealed that in the vast majority of cases the date to be entered is close to the current date and has a tendency to be in the future rather than the past.

Accordingly this design is optimised for dates in the twenty first rather than the twentieth, or other, century. In its initial state it shows the year 2000 and as shown this is the only state where a delete key press will occasion the history transition back into the month entry part. Should 2000 be the year required, or should the year required be shown in any of the subsequent year entry states, pressing the <enter> key will cause the transition to the terminal state, but only if leap year considerations are satisfied. The terminal state is distinguished from the year entry states by the inclusion of the day of week, as shown in the last part of Figure 2.

From the 2000 state any of the digit key presses '1' to '9' will cause a transition to the state identified as 200W. Hence any of the years 2001 to 2009 can be selected with a single key press. A further key press, this time of any digit '0' to '9', will allow any of the years in the range 2010 to 2099 to be defined, identified as 20WX. For example if the year required was 2045, from the 2000 state pressing '4' would display 2004 and then pressing '5' would display 2045 following which the <enter> key could be pressed to confirm it.

There are two further transitions occasioned by the '1' to '9' keys leading to the 2WXY and subsequently to the WXYZ states. Hence any year prior to 2000 or beyond 2999 would require the maximum of four key presses, whereas those in the current millennium would require a maximum of 3 presses and those in the current century a maximum of 2. It is likewise optimised for the current decade a situation that will not last for long.

Caveats and considerations

The design can be readily implemented, a fragment of the Java implementation of the month entry part is given in Figure 6. The full source code, a working implementation and a copy of the statechart is available at (Culwin 2003). The source code can also be used as part of the case study to illustrate how a complete design can inform an implementation. For example, the design compartmentalises the input of the three parts of the date and this compartmentalisation is reflected in three methods that implement each part of the design.

```

case JAN:
    if ( characterPressed == '1' &&
        dayNumber != 31 ){
        newState = NOV;
        monthString.append( characterPressed);
    } else if ( characterPressed == '0' ||
               characterPressed == '2' ){
        newState = OCT_OR_DEC;
        monthString.append( characterPressed);
    } else if ( virtualKeyPressed == KeyEvent.VK_BACK_SPACE) {
        newState = lastDayState;
        monthString = null;
    } // End if.
    break;

case MAR_OR_MAY_OR_JLY_OR_AUG:
case APR_OR_JUN_OR_SPT:
case FEB:
    if ( virtualKeyPressed == KeyEvent.VK_BACK_SPACE) {
        newState = lastDayState;
        monthString = null;
    } // End if.
    break;

```

Figure 6 Implementation fragment

The widget as presented implements the complete design as presented but it is not yet developed to the point where it could be deployed. As a Java component that extends the JTextField class it inherits many attributes and methods that will allow it to inter-operate with other Java components. However it lacks an inquiry method which would allow the date that it is displaying to be obtained and should also have the capability to dispatch a ChangeEvent when it follows the final transition. It would also best be packaged as a JavaBean.

There is one anticipated usability problem with the design as presented. It assumes that the user will complete the entire interaction, however it can be anticipated that this will not always be the case. A user may transfer focus from the component at any stage up to and including the year entry stage. This would leave the date only partially entered and the subsequent behaviour of the component were the inquiry method to be used needs to be defined. The safest decision might be to throw an exception should the component not be in one of the year entry states. To facilitate this the final transition could be amended to include a change of focus event as well as a delimit key event.

One further refinement that might be considered would be to supply a default date, possibly today's date depending

upon the precise requirement, in the initial state and have a terminal transition connected to that state. This would allow the user to confirm the default simply by using the <enter> key.

Empirical side by side usability investigation of this component has not been possible. An attempt was made in the study reported in (Culwin & Faulkner in prep) but the users allocated to use this component were observed to be intrigued by its behaviour and spent an inordinate amount of time playing with it and exploring its behaviour. Consequently the time taken to complete the task could not be compared with other mechanisms. Additionally, as the users were attending to its behaviour and state much more closely than the other, more familiar, mechanisms the data related to correct task completion was regarded as suspect and not included in the study. Observationally the users reported greater enjoyment using this mechanism, but this could well be a Hawthorne effect.

Conclusion

This paper has presented the design and made available the implementation of a novel date entry mechanism. It is hoped that if it were deployed it might obviate some of the known problems with existing mechanisms. It has also provided an introductory case study that demonstrates the power and utility of statechart design notations. The case study differs from other common pedagogic examples by avoiding the complexity of real time systems or of windowing systems, whilst providing a software example of moderate complexity. It also allows the full design to be revealed in three stages allowing its complexity to be managed. As an introductory example it presents the essence of statechart notation but other, more complex aspects such as concurrent statecharts, are omitted for subsequent examples to introduce.

Acknowledgements

A group of final year computing students led by Mohammed Hussain designed, built and empirically evaluated a behaviourally identical initial version of this mechanism.

References

- Bainbridge, A., (2002), Hotel date entry design & usability, http://www.travelucd.com/research/pdf/date_entry_hotel_july2002.pdf, (15 Oct 2003).
- Culwin, F., (2003), Intelligent Date Input Mechanism, <http://myweb.lsbu.ac.uk/~fintan/idim/idim.html>, (15 Oct 2003).
- Culwin, F., (2000), *A Java Foundation Classes Primer*, Macmillan ISBN 0-333-77339-X.
- Culwin, F., (1998), *A Java GUI Programmer's Primer*, Prentice Hall ISBN 0-13-908849-0.
- Culwin, F., (1994), *An X/Motif Programmer's Primer*, Prentice Hall ISBN 0-13-101841-8.
- Culwin, F., and Faulkner, X., (in prep), Date entry a schema, survey and empirical investigation, to appear.
- Grigg, A., and Henderson, N., (2000), A Systematic Method for Development of Real-Time Systems, Department of Computing Science, University of Newcastle, <http://www.cs.ncl.ac.uk/research/pubs/trs/papers/710.pdf> (15 Oct

2003).

Harel, D., (1988), On visual formalisms, *Communications of the ACM*, 31(15), pp514-530.

Horrocks, I., *Constructing the User Interface with Statecharts*, Addison Wesley; ISBN: 0201342782.

Köhler, H.J., Nickel, U., Niere, J., and Zündorf A., (2000), Integrating UML diagrams for production control systems, *Proc.22nd ACM international conference on Software engineering*, Pages: 241 - 251.

Rumbaugh, J., Jacobson, I., and Booch, G., (1999), *Unified Modelling Language Reference Manual*, Addison Wesley; ISBN: 020130998X.

Samek, M., (2002), *Practical Statecharts in C/C++*, CMP Books 1578201101.