Preprint: Final version published as
BLANDFORD, A., GREEN, T. & CONNELL, I. (2005) Formalising an understanding of user–system misfits. In R. Bastide, P. Palanque & J. Roth (Eds.) *Proc. EHCI-DSVIS 2004*. Springer: LNCS 3425. 253-270

Formalising an understanding of user–system misfits

Ann Blandford[1], Thomas R. G. Green[2] and Iain Connell[1]

[1] UCL Interaction Centre, University College London, Remax House, 31-32 Alfred Place
London WC1E 7DP, U.K.
{A.Blandford,I.Connell}@ucl.ac.uk
http://www.uclic.ucl.ac.uk/annb/
[2] University of Leeds, U.K.

Abstract. Many of the difficulties users experience when working with interactive systems arise from misfits between the user's conceptualisation of the domain and device with which they are working and the conceptualisation implemented within those systems. We report an analytical technique called CASSM (Concept-based Analysis for Surface and Structural Misfits) in which such misfits can be formally represented to assist in understanding, describing and reasoning about them. CASSM draws on the framework of Cognitive Dimensions (CDs) in which many types of misfit were classified and presented descriptively, with illustrative examples. CASSM allows precise definitions of many of the CDs, expressed in terms of entities, attributes, actions and relationships. These definitions have been implemented in Cassata, a tool for automated analysis of misfits, which we introduce and describe in some detail.

1 Introduction

Two kinds of approach have dominated traditional work in usability of interactive systems: *heuristic* (or *checklist-based*) approaches giving a swift assessment of look-and-feel (usually independent of the tasks the system is designed to support), such as Heuristic Evaluation [17]; and *procedure-based* approaches for assessing the difficulty of each step of typical user tasks, such as Cognitive Walkthrough [20].

We present a technique based on a third approach, the analysis of *conceptual misfits* between the way the user thinks and the representation implemented within the system. Such misfits pertain to the concepts and relationships the user is manipulating in their work. Some misfits are surface-level – for example, users may work with concepts that are not directly represented within the system; conversely, users may be required to discover and utilise system concepts that are irrelevant to their conceptual models. Other misfits are structural, emerging only when the user manipulates the

structure of some representation and finds that changes that are conceptually simple are, in practice, difficult to achieve.

We outline an approach to usability evaluation called Concept-based Analysis of Surface and Structural Misfits (CASSM), and present Cassata, a prototype analysis tool that supports the analyst in identifying misfits. As will become apparent, in CASSM structural misfits are not analysed directly in terms of the procedures that users follow to make a change, as might happen using a procedural approach; instead, CASSM identifies which elements of a structure are and are not accessible to a user and amenable to direct modification, thereby deriving warnings of potential misfits.

1.1 Misfits and their analysis

Many approaches to usability evaluation, including work in the previously-mentioned traditions of heuristic and procedure-based analysis, have generated lists of specific user problems with a given design, but have failed to impose any structure on the lists. Each user difficulty that is spotted is a thing in itself. From one occurrence we learn nothing about how to predict further occurrences, nor how to improve design practice.

CASSM builds on the approach known as the 'Cognitive Dimensions of Notations' framework (CDs) [3,4,14,15], in which some important classes of structural misfits have been articulated and described. For example, 'viscosity' describes the 'degree of resistance to small changes': in a viscous system, something is more difficult to change than it should be – a single conceptual action demands several device actions. An example would be adding a new figure near the beginning of a document then having to increment all subsequent figure numbers and within-text references to those numbers: some word processing applications explicitly support this activity but most do not, making it very repetitive. Viscosity may be a serious impediment to the user's task or it may be irrelevant to that task, if for instance the user is searching for a target but not trying to make a change; the CDs framework therefore distinguishes types of user activity and offers a conjecture as to how each dimension affects each activity.

The Cognitive Dimensions framework as originally created [12] was intended to promote quick, broad-brush evaluation, giving non-specialists a usability evaluation technique that was based on cognitive analysis yet required no expertise from the analyst. It relied purely on definition by example. To a degree this was successful. The idea of viscosity is intuitively appealing; examples can illustrate the idea; and a vocabulary of such ideas can be used to support discourse and reasoning about features of a design, with a view to improving that design [3]. However, despite the development of a CDs tutorial [14], and a questionnaire-based evaluation tool [2], potential users have found that they need to learn too many concepts and that those concepts are not defined closely enough to avoid disagreement over the final analysis.

More than one attempt has been made to sharpen the definitions of CDs [11,19] but those attempts have lost the feel of quick, broad-brush evaluation, making them unappealing to the intended user, the non-specialist analyst.

In this paper, we show that several CDs and related user–system misfits can be represented reasonably faithfully in a form that better preserves the original quick-and-dirty appeal of CDs. With these definitions, not only are the misfit notions

clarified, but it becomes possible for potential misfit occurrences to be automatically identified within Cassata, the tool that we shall describe below.

It must be kept in mind throughout that our form of analysis can only describe *potential* user problems. Whether a particular misfit causes real difficulties will depend on circumstances that are not described here.

2 CASSM and Cassata: a brief introduction

CASSM is a usability evaluation technique that focuses on the misfits between user and device. It was formerly known as Ontological Sketch Modelling (OSM [10]), because the approach involves constructing a partial (Sketchy) representation (Model) of the essential elements (Ontology) of a user–system interaction; the name has recently been changed to reflect a shift of focus towards the two types of misfits rather than the ontology representation.

CASSM developed from our earlier work on Entity Relationship Modelling of Information Artifacts (ERMIA [11]) and Programmable User Modelling (PUM [8]). It has also been informed by the work of others on what could broadly be termed misfit analysis, such as Moran's External Task Internal Task (ETIT) analysis [16] and Payne's Yoked State Spaces [18]. The basis of CASSM is to compare the concepts that users are working with (identified by an appropriate data gathering technique such as interviews, think-aloud protocols or Contextual Inquiry [1]) with the concepts implemented within the system and interface (identified by reference to sources such as system documentation or an existing implementation). Conceptual analysis involves identifying the concepts users are working with, drawing out commonalities across similar users (see for example [7]) to create the profile of a typical user of a particular type,; the analyst can then assess the quality of fit between user and system. As analysis proceeds, the analyst will start to distinguish between entities and attributes (as defined below), and to consider what actions the user can take to change the state of the system. Finally, for a thorough analysis, various relationships between concepts are enumerated to identify structural misfits. Each of these stages of misfit analysis is discussed in more detail below.

To support analysis, a demonstrator tool called Cassata is under development. Screen shots included in this paper are taken from version 2.1 of the tool. (Version 3 can be downloaded from the project web page [9].) The tool has provided a focus for developing the precise definitions of misfits included in this paper, and also a means of testing those definitions against a repertoire of examples that have previously been discussed informally.

Figure 1 shows the Cassata window for a partial description of a word processor document. For clarity, the picture is cropped from the right. This particular description is discussed in more detail in section 4.1; here we simply outline its main features.

It is a description of a set of figures (pictures or diagrams) in a document, which consists of one or more individual figures. For the user, there is the important idea that the figures should be sequentially numbered – so the *number-sequence* is important, and is an attribute of the *set-of-figs*. Each *figure* has an attribute which is

its particular *number*, and changing a figure number changes the overall sequence of figure numbers.

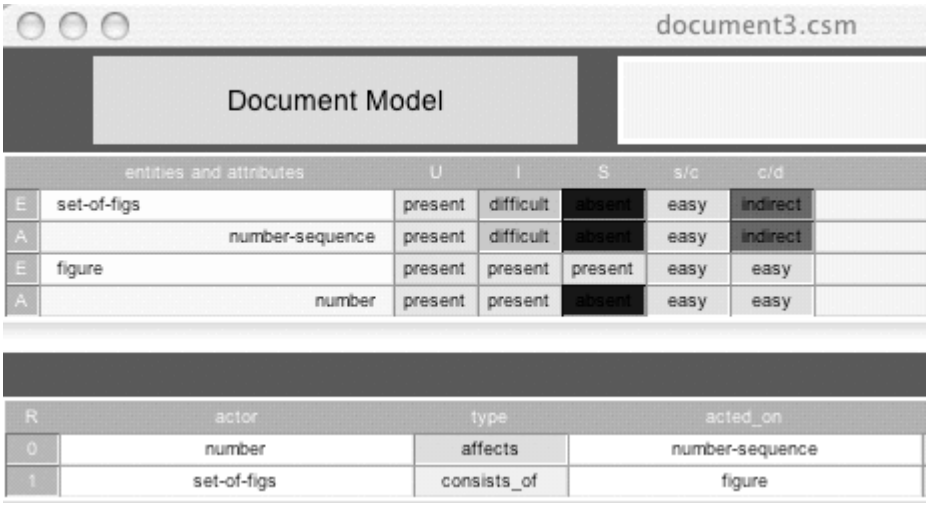

Fig. 1. Cassata data table for a partial description of a document. The upper table describes concepts (i.e. entities and their attributes); the lower describes relationships between those concepts.

The top half of the window shows information about concepts (entities such as *figure* and attributes such as *number*): for each concept, three columns show whether it is present, difficult or absent for the user, interface and system respectively; the next two columns show how easy it is to set or change the value of an attribute, or to create or delete an entity; the final column is a notes area in which the analyst can add comments. To take the first row as an example: the `set-of-figs` is a conceptual entity that is meaningful to the user, is not clearly represented at the interface ('difficult') and absent from the underlying system model. It is easy to create a set of figures, (because this happens automatically as the user adds figures) but harder to delete it (done indirectly because that requires deleting *all* the individual figures).

The bottom half of the window shows information about relationships (such as *affects* and *consists_of*) between concepts. In this particular case, the two lines of input state that changing any *number* (of a *figure*) affects the *number-sequence* (of the *set-of-figs*) and that a *set-of-figs* consists of (many) *figures*.

Having briefly presented the background to CASSM and Cassata, we now focus in more detail on the definitions of various kinds of misfits.

3 Surface Misfits

Surface misfits are those that become apparent without considering the details of structural representations within the system and how those representations are changed. Within 'surface', there are three levels of misfit: just identifying system and

user concepts, with little reference to the interface between the two (section 3.1); more detailed analysis in terms of how well each concept is represented by the user, interface and system (section 3.2); and analysis in terms of what actions are needed to change the system, and whether there are problems with actions (section 3.3).

3.1 Level 1: Misfits between the user and the system

Misfits between user and system are probably the most important surface-level misfits. There are three important cases: user concepts that are not represented within the system; system concepts that are inaccessible to the user; and situations where a user concept and a system concept are similar but not identical.

User concepts that are not represented within the system cannot be directly manipulated by the user. The *set-of-figs* discussed above is an example of such a concept. Other examples are using a field in an electronic form to code information for which that form was not actually designed, or keeping paper notes alongside an electronic system to capture information that the system does not accept.

Unrepresented concepts are often the most costly form of misfit; they may force users to introduce workarounds, as users are unable to express exactly what they need to, and must therefore use the system in a way it was not designed for. They sometimes result in structural misfits such as viscosity, as described below.

System concepts that are not immediately available to the user need to be learned. At a trivial level, these might include strictly device-related concepts like scroll-bars, which may be simple to use but nevertheless need to be learnt. A slightly more complex example is the apparatus of layers, channels and masks found in many graphics applications – these can cause substantial user difficulties, particularly for novice users.

For users, these misfits may involve no more than learning a new concept, or they may require the users' constant attention to the state of something that has little significance to them, such as the amount of free memory.

User- and system concepts that are similar but non-identical, and which are often referred to by the same terms, can cause more serious difficulties. One example in the domain of diaries is the idea of a 'meeting'. When a user talks about a meeting, they usually mean a pre-arranged gathering of particular individuals at an agreed location with a particular broad purpose (and perhaps a detailed agenda). Within some shared diary systems, a meeting has a much more precise definition, referring to an event about which only other users of the same shared diary system can be kept fully informed, and which has a precise start time and precise finishing time, and possibly a precise location. The difference between these concepts is small but significant [5].

Another example, within the domain of ambulance dispatch, is the difference between a *call* and an *incident*. A particular system we studied processed information strictly in terms of calls, whereas staff dealt with incidents (about which there may be one or many calls); this was difficult to detect initially because the staff referred to

them as ‘calls’ [7], but the failure of the system to integrate information about difference calls added substantially to staff workload as they processed the more complex incidents.

These misfits may cause difficulties because the user has to constantly map their natural understanding of the concept onto the one represented within the system, which may have a subtly different set of attributes.

3.2 Level 2: Adding Interface Considerations

The second level starts to draw out issues concerning the interface between user and system. For each of user, interface and system, a concept may be *present*, *difficult* or *absent*.

In all cases, *present* means clearly represented and *absent* means not represented. We assume that underlying system concepts are either present or absent, whereas for the user or at the interface there are concepts that are present but *difficult* in some way.

For users, *difficult* concepts are most commonly ones that are implicit– ideas they are aware of if asked but not ones they expect to work with. An example would be the end time of a meeting in the diary system mentioned above: if one looks at people’s paper diaries, one finds that many engagements have start times (though these are often flagged as approximate – e.g. ‘2ish’) but few have end times, whereas electronic diaries require every event to have an end time. This forces users to make explicit information that they might not choose to. Another source of difficulty might be that the user *has to learn* the concept.

Similarly, there are various reasons why a concept may be represented at the interface but in a way that makes it difficult to work with. Difficulties that interface objects may present include:

- *Disguised*: represented, but hard to interpret;
- *Delayed*: represented, but not available to the user until some time later in the interaction;
- *Hidden*: represented, but the user has to perform an explicit action to reveal the state of the entity or attribute; or
- *Undiscoverable*: represented only to the user who has good system knowledge, but unlikely to be discovered by most users.

Which of these apply in any particular case – i.e. why the interface object might cause user difficulties – is a further level of detail that can be annotated by the analyst; for the sake of simplicity, this additional level of detail is not explicitly represented within Cassata.

At the simplest level, anything that is *difficult* or *absent* represents a misfit that might cause user difficulties. As discussed above, concepts that are difficult or absent for the user are ones that need to be learnt and worked with; how much difficulty these actually pose will depend on the interface representation. Conversely, concepts that are present for the user but absent from the underlying system will force the user to find work-arounds. In addition, as discussed above, poor interface representations are a further source of difficulty that are not considered at level 1.

3.3 Level 3: Considering Actions

At levels 1 and 2, we have referred to 'concepts' without it being necessary to distinguish between them. For deeper analysis, it becomes necessary to distinguish between entities and attributes. A description in terms of entities and attributes is illustrated in the screen-shot from the Cassata tool shown in Figure 1 (above). There, we used the terms 'entity' and 'attribute' without precisely defining them.

An *entity* is a concept that can be created or deleted, or that has attributes which the analyst wants to enumerate. In figure 1, entities are shown in the left-hand column, left-justified. Note also the 'E' in the left margin.

An *attribute* is a property of an entity. In Figure 1, attributes are shown right-justified in the left-hand column. Note also the 'A' in the left margin. Attributes can be set ('**S**/C') or changed ('**C**/D').

For economy of space, the same columns are used to define how easy it is to create ('S/**C**') or delete ('C/**D**') entities. Each of these actions can be described as follows:

- *Easy*: no user difficulties.
- *Hard*: difficult for some reason (e.g. undiscoverable action, moded action, delayed effect of action). For example, it is possible to select a sentence in MS Word by pressing the control key ('apple' key on a Mac) and clicking anywhere in the sentence; few users are aware of this.
- *Indirect*: effect has to be achieved by changing something else in the system; for example, as discussed above, it is not possible to directly change the sequence of figure numbers.
- *Cant*: something that cannot be changed, that the analyst thinks might cause subsequent user difficulties.
- *Fixed*: something that cannot be changed, that is not, in fact, problematic; for example, an entity may be listed simply because it has important attributes that need to be enumerated or analysed.
- *BySys*: this denotes aspects of the system that may be changed, but not by the user (this may include by other agents – e.g. over a network, or simply other people). Many of these cases are not actually problems, and it is up to the analyst to consider implications.

Just as describing concepts as 'present', 'absent' or 'difficult' helps to highlight some conceptual difficulties, so describing actions in terms of 'easy', 'hard' , indirect', 'cant', 'fixed' and 'bySys' highlights conceptual difficulties in changing the state of the system.

3.4 Surface-level misfits and their Cognitive Dimensions

We turn now to the use of CASSM to articulate part of the Cognitive Dimensions framework introduced above, starting with surface-level misfits – notably abstraction level and visibility.

Abstraction level: devices may be classed as imposing the use of abstractions ('abstraction-hungry' in Green's terminology), rejecting the use of abstractions ('abstraction-hating'), or allowing but not imposing abstractions ('abstraction-neutral'); further, the abstractions themselves may be domain-based or device-based. CASSM can express these distinctions reasonably well and can therefore detect some of the misfits, among them:

- domain abstractions that are part of the user's conceptual but are not implemented within the device;
- device abstractions imposed upon the user.

Imposed device abstractions have to be learnt in order to work effectively with the device, such as style sheets or graphics masks, and are therefore easy or difficult to learn according to how well they are represented at the interface (as discussed above).

Visibility: the user's ability to view components readily when required, preferably in juxtaposition to allow comparison between components. CASSM cannot at present express either inter-item juxtaposability nor the number of search steps required to bring a required item to view ('navigability') but captures the essence of visibility by designating those concepts that are hidden, disguised, delayed or undiscoverable as 'difficult' in the interface representation.

4 Structural misfits: taking account of relationships

As discussed above, structural misfits refer to the structure of information, and how the user can change that structure. Here, we present the structural misfits of which we are currently aware. These are a subset of Green's Cognitive Dimensions [3]. It is worth noting that structural misfits only apply to systems where the system state can be changed in a meaningful way by the user. Thus, systems such as web sites or vending machines do not generally suffer from structural misfits. However, systems such as drawing programs, word processors, music composition systems and design tools are prone to these misfits.

Another point to note is that although structural misfits are much finer-grained than the bolder surface-level misfits discussed above, they can be immense sources of user frustration and inefficiency.

Structural misfits depend on relationships that hold within the data. Five kinds of relationships are currently defined within Cassata. These are: *consists_of*, *device_constraint*, *goal_constraint*, *affects*, and *maps_onto*. As for entities and attributes, it is possible (though not always necessary) to state how well these relationships are represented at the interface, to the user, or in the underlying system.

Consists_of takes two arguments, which we call Actor and ActedOn, which are both concepts. This means that the first consists_of the second: chapter consists_of paragraphs; set-of-paragraphs consists_of paragraphs (e.g. sharing a paragraph style); etc.

Device_constraint also takes two arguments, both concepts. The value of Actor constrains the possible values of ActedOn. For example, considering drawing a map on the back of an envelope, the starting_position (for drawing) constrains the

location of a particular instruction. An easier example is that the field-width for a data entry field constrains the item-width for any items to be put in that field.

Goal_constraint takes only one argument (ActedOn), which is the concept on which there is some domain-based constraint. For example, when writing a conference paper such as this one, it is common to have a limit on the length of a document.

Affects is concerned with side-effects: that changing the value of one concept will also change the value of another. For example, changing the number of words in a document will change its length.

Maps_onto is a simple way of expressing the idea that two concepts are very similar but not quite identical. These are most commonly a domain-relevant concept and a device-relevant one. For example, a (user) meeting maps_onto a (diary-entry) meeting but, depending on the form of the diary, the two meeting types may have importantly different attributes.

We now consider three important classes of structural misfits: viscosity (section 4.1), premature commitment (section 4.2) and hidden dependencies (section 4.3). In what follows, we take A to be an entity of interest with an attribute P, and B to be some other entity with attribute Q. these are defined in the top window by juxtaposition (i.e. attributes always appear immediately below the entity to which they pertain).

4.1 Viscosity

As discussed above, "viscosity" captures the idea that a system is difficult to change in some way. Green [13] distinguished two types of viscosity, repetition and knock-on, which can be defined as follows.

1) Repetition viscosity occurs when a single action within the user's conceptual model requires many, repetitive device actions.

Changing attribute P of entity A, A(P), needs many actions if:

```
A(P) is not directly modifiable
B(Q) affects A(P)
B(Q) is modifiable
A consists-of B
```

For example, as discussed above (section 2), we get repetition viscosity on figure numbers in a document because whenever a figure is added, deleted or moved, a range of figures need to be re-numbered one by one. Stated more formally:

```
set-of-figs(number-sequence) is not directly modifiable
figure(number) affects set-of-figs(number-sequence)
figure(number) is modifiable
set-of-figs consists-of figure
```

Figure 2 shows the basic requirements on a model for it to exhibit Repetition Viscosity. Note in particular the use of 'indirect' to denote something that can be

changed, but not directly. Figure 3 shows the output when this particular model is assessed by Cassata.

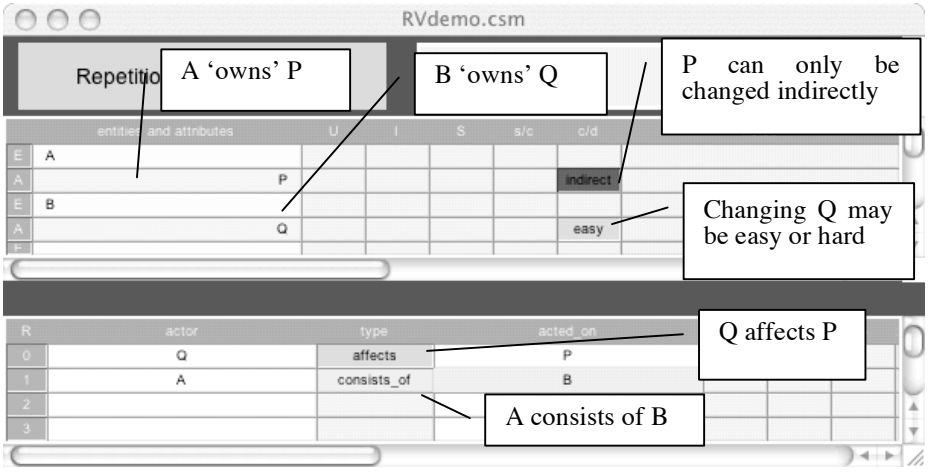

Fig. 2. Repetition Viscosity

```
Repetition Viscosity Check ---- Repetition Viscosity Model

  attribute "Q" affects "P"
  entity "A" consists_of "B"
 "A " owns "P"
 "P " is not directly modifiable
 "B " owns "Q"
 "Q " is directly modifiable

possible case of repetition viscosity:
to change "P" user may have to change all instances of "Q"
```

Fig. 3. Output from Repetition Viscosity analysis in Cassata

2) Knock-on viscosity: changing one attribute may lead to the need to adjust other things to restore the internal consistency. (In North America, a better-known phrase for the same concept appears to be 'domino effect'.)

Changing A(P) has possible knock-on if:

```
A(P) is modifiable
modifying A(P) affects B(Q)
there is a domain constraint on B(Q)
```

Timetables and schedules typically contain high knock-on viscosity; if one item is re-scheduled, many others may have to be changed as well.

Figure 4 shows the conditions for a model to exhibit Knock-on Viscosity. Figure 5 shows the output when this model is assessed by Cassata.

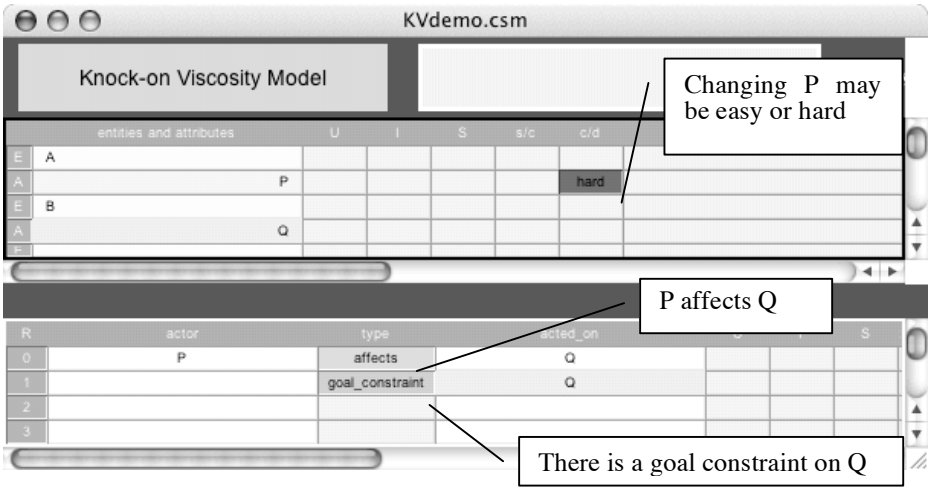

Fig. 4. Knock-on Viscosity

```
Knock-on Viscosity Check ---- Knock-on Viscosity Model

  attribute "P" affects "Q"
  there is a goal_constraint on "Q"
 "P " is directly modifiable

possible case of knock-on viscosity
modifying "P" may violate a domain constraint for "Q"
```

Fig. 5. Output from Knock-on Viscosity analysis in Cassata

4.2 Premature Commitment

Informally, premature commitment occurs when the user has to provide information to the system earlier than they would wish or are prepared for. We have several sets of conditions that alert to possible premature commitment.

1) Non-modifiability premature commitment: As discussed above (under actions), if an attribute cannot be changed after it has been set then the system possibly demands premature commitment:

```
A(P) is settable
A(P) is not modifiable
```

Some painting tools exhibit this type of premature commitment: that the width and colour of a line cannot be changed once it has been set.

Extending this to entities, we may get potential non-modifiability premature commitment if entities can be created but not subsequently deleted:

```
A is creatable
A is not deletable
```

In principle the converse may hold too, but there are few situations in which that would class as premature commitment (rather than simply an irreversible action).

Figure 6 shows the conditions for a model to exhibit this kind of Premature Commitment. Figure 7 shows the output when this particular model is assessed by Cassata.

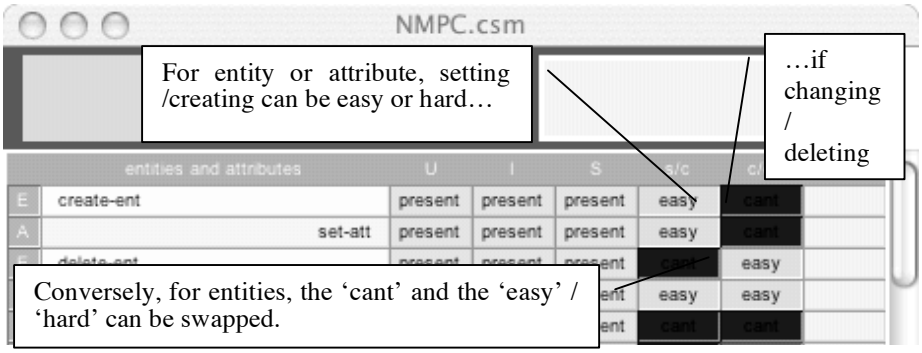

Fig. 6. Non-modifiability Premature Commitment

```
Non-modifiability Premature Commitment ---- test NMPC Model

possible non-modifiable premature commitment:
  entity "create-ent" can be created but not deleted
====

possible non-modifiable premature commitment:
  attribute "set-att" can be set but not changed
====

possible non-modifiable premature commitment:
  entity "delete-ent" can be deleted but not created
====
```

Fig. 7. Output from Non-modifiability PC analysis in Cassata

2) Abstraction-based premature commitment: If a user has to define an abstraction in order to avoid repetition viscosity, and that abstraction has to be defined in advance, then the system potentially creates abstraction-based premature commitment. Frequently that abstraction will be a simple grouping. A common example of potentially premature commitment to abstractions is the defining of paragraph styles before starting to create a technical document. The purpose is to avoid repetition viscosity by allowing all paragraphs of one type to be reformatted in one action, but the problem is to foresee the required definitions. A more technical example would be the creation of a class hierarchy in object-oriented programming.

The conventional analysis in the Cognitive Dimensions framework is to treat the abstraction management components of the system as a separate sub-device, which may have its own properties of viscosity, hidden dependencies, etc [4]. In CASSM we take a simplified approach such that this type of premature commitment is highlighted if:

```
A consists-of B
A(P) is directly modifiable
A(P) affects B(Q)
```

The paragraph styles case would be represented thus:

```
Paragraph has attribute style
Set-of-paragraphs has attribute style-description
Set-of-paragraphs consists-of paragraph
Style-description is directly modifiable
Changing style-description causes style to change
```

Figure 8 shows the basic requirements on a model for it to exhibit Abstraction-based Premature Commitment. Figure 9 shows the output when this particular model is assessed by Cassata.

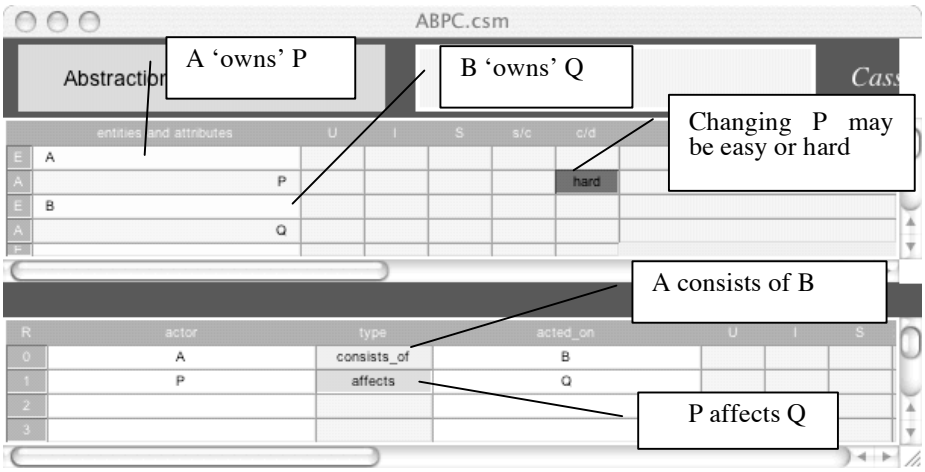

Fig. 8. Abstraction-based premature commitment

```
Abstract-based Premature Commitment Check ---- Abstraction-based PC Model

  attribute "P" affects "Q"
  entity "A" consists_of "B"
 "A " owns "P"
 "P " is directly modifiable
 "B " owns "Q"

possible case of abstract-based premature commitment:
need to create an abstraction "A" to change all instances of "Q"
```

Fig. 9. Output from Abstraction-based PC analysis in Cassata

3) Device-constraint premature commitment: Here, setting an attribute of one entity constrains the way that new instances of another entity can be created:

```
B(Q) is settable
A(P) is not settable
There is a device constraint between B(Q) and A(P)
It is possible to add more As
```

As mentioned above (when defining *device constraint*), one example of this is drawing a map on the back of an envelope; another is that of setting the field width in a data structure when the size of all items to be entered in that field is not known (here, ">=" is an example of a device constraint):

```
field(width) is settable
item(width) is not settable
field(width)>=item(width)
more items can be added
```

Figure 10 shows the basic requirements on a model for it to exhibit Device-constraint Premature Commitment. Figure 11 shows the output when this particular model is assessed by Cassata.

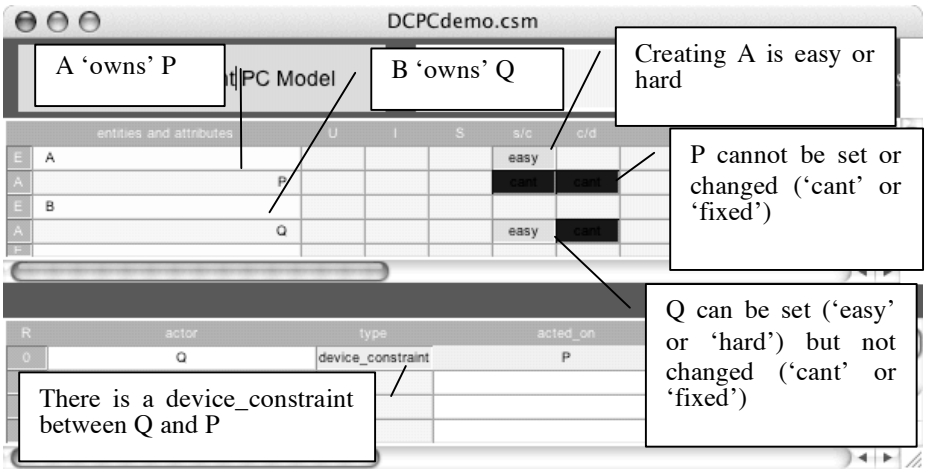

Fig. 10. Device-constraint premature commitment

Device-constraint Premature Commitment Check ---- Device-constraint PC Model

attribute "Q" imposes a device_constraint on "P"
"Q " can be set but not changed
"P " cannot be either set or changed
"A " can be created

possible case of device-constraint premature commitment:
attribute "P" may be constrained by "Q"

Fig. 11. Output from Device-constraint PC analysis in Cassata

4.3 Hidden Dependencies

A hidden dependency occurs when important links between concepts are not visible (or otherwise readily available to the user). Spreadsheets contain many hidden dependencies, so that changing a value or formula somewhere in a sheet can have unanticipated knock-on effects elsewhere in the sheet. Similarly, changing a style in MS Word can have unexpected knock-on effects on other styles through the style hierarchy. This is formalised simply:

```
Changing C affects D
The relationship is not visible
```

Here, C and D are concepts (entities or attributes). They may even be the same concept. For example, in the word processor because the concept 'style definition' denotes an aggregate of styles formed into a hierarchy, changing any one definition potentially changes other definitions that refer to it, so we have the reflexive relationship:

```
Changing style-definition affects style-definition
The relationship is not visible
```

Figure 12 shows the basic requirements on a model for it to exhibit Hidden Dependencies. Figure 13 shows the output when this particular model is assessed by Cassata.

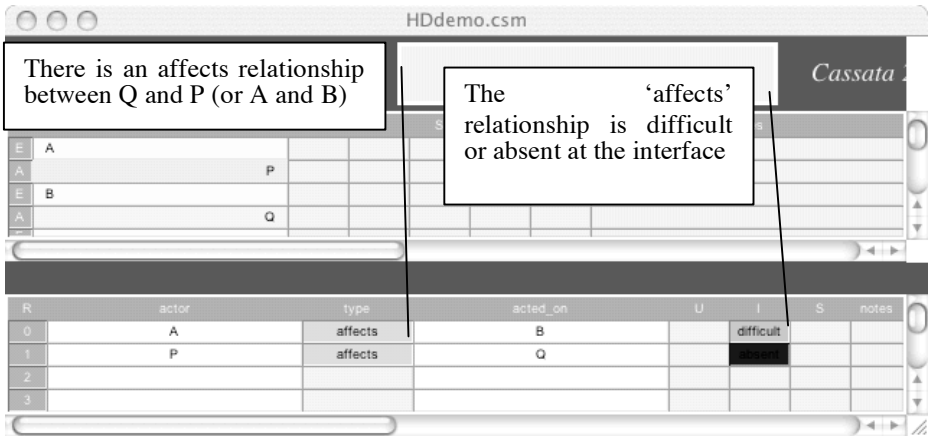

Fig. 12. Hidden Dependencies

```
Hidden Dependencies Check ---- Hidden Dependencies Model

  "A" affects "B"

possible case of hidden dependency:
there may be hidden dependency between "A" and "B"
====
  "P" affects "Q"

possible case of hidden dependency:
there may be hidden dependency between "P" and "Q"
```

Fig. 13. Output from Hidden Dependencies analysis in Cassata

5 Conclusions

In this paper, we have presented a particular approach to assessing the usability of an interactive system based on the idea of 'quality of fit' between user and system. In particular, we have used the ontology of CASSM (considering entities, attributes, actions and a set of defined relationship types, and properties of each of these) to deliver precise definitions of various kinds of surface and structural misfits. The structural misfits are all based on Green's [12] Cognitive Dimensions. Some of the surface misfits can also be identified as CDs, but most are not, and all have been independently derived from the basic CASSM ontology.

The prototype Cassata tool allows CASSM-based descriptions of systems to be created quickly and with a minimum of special concepts. When a CASSM description has been entered into Cassata, potential occurrences of both surface and structural misfits can be automatically identified, thereby alerting analysts to possible usability problems. With the help of Cassata we have preserved the original quick-to-do feel of

the Cognitive Dimensions analysis, unlike previous efforts at formalising the Cognitive Dimensions framework [11,19].

In practice, we have found that it is usually easier to identify structural misfits informally (as has been done historically with CDs) than by generating the full CASSM representation in Cassata; in this case, the role of the formalisation is to validate that informal understanding and make it more precise. The Cassata tool provides simple but valuable support in identifying both surface and structural misfits.

We are not claiming that the set of misfits presented here is complete. There are many different *kinds* of misfits between users and systems, many of which are outside the scope of CASSM – for example, inconsistencies in procedures for similar tasks would be picked up by other techniques but are not directly addressed within CASSM. In this work, we have focused on conceptual misfits, which have not been widely recognised in earlier work on usability evaluation.

The work reported here is ongoing; elsewhere, we have reported the application of CASSM to various kinds of interactive systems [7,10]. Current work is addressed at refining the Cassata prototype, extending the set of structural misfits and scoping CASSM by comparison with other usability evaluation techniques (e.g. [6]). We believe that this work makes an important contribution to the overall repertoire of evaluation approaches for interactive systems.

Acknowledgements

This work is supported by EPSRC grant GR/R39108.

References

1. Beyer, H., Holtzblatt, K.: *Contextual Design*. San Francisco : Morgan Kaufmann. (1998).
2. Blackwell, A.F., Green, T.R.G.: A Cognitive Dimensions questionnaire optimised for users. In A.F. Blackwell & E. Bilotta (Eds.) *Proceedings of the Twelfth Annual Meeting of the Psychology of Programming Interest Group* (2000).137-152.
3. Blackwell, A., Green, T. R. G.: Notational systems – the Cognitive Dimensions of Notations framework. In J. Carroll (ed.), *HCI Models, Theories and Frameworks*, Morgan Kaufmann. (2003) 103-134.
4. Blackwell, A., Hewson, R., Green, T. R. G.: The design of notational systems for cognitive tasks. E. Hollnagel (ed.) In E. Hollnagel (Ed.), *Handbook of Cognitive Task Design*. Mahwah, N.J.: Lawrence Erlbaum. (2003) 525-545.
5. Blandford, A. E., Green, T. R. G.: Group and individual time management tools: what you get is not what you need. *Personal and Ubiquitous Computing*. Vol 5 No 4. (2001) 213–230.
6. Blandford, A., Keith, S., Connell, I., Edwards, H.: Analytical usability evaluation for Digital Libraries: a case study. In *Proc. ACM/IEEE Joint Conference on Digital Libraries*. (2004) 27-36.
7. Blandford, A. E., Wong, B. L. W., Connell, I. W., Green, T. R. G.: Multiple viewpoints on computer supported team work: a case study on ambulance dispatch. In X. Faulkner, J. Finlay & F. Détienne (eds), *Proc. HCI 2002 (People and Computers XVI)*, Springer (2002) 139-156.

8. Blandford, A. E., Young, R. M.: Specifying user knowledge for the design of interactive systems. *Software Engineering Journal*. 11.6, (1996) 323-333.
9. CASSM: Project web site www.uclic.ucl.ac.uk/annb/CASSM.html
10. Connell, I., Green, T., Blandford, A.: Ontological Sketch Models: highlighting user-system misfits. In E. O'Neill, P. Palanque & P. Johnson (Eds.) *People and Computers XVII, Proc. HCI'03*. Springer. (2003) 163-178.
11. Green, T. R. G., Benyon, D.: The skull beneath the skin: entity-relationship models of information artifacts. *International Journal of Human-Computer Studies*, 44 (1996) 801-828
12. Green, T. R. G.: Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay (Eds.) *People and Computers V*. Cambridge University Press. (1989) 443-460
13. Green, T.R.G.: The cognitive dimension of viscosity - a sticky problem for HCI. In D. Diaper and B. Shackel (Eds.) *INTERACT '90*. Elsevier. (1990)
14. Green, T. R. G., Blackwell, A. F.: Cognitive dimensions of information artefacts: a tutorial. http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf (1998)
15. Green, T. R. G., Petre, M.: Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *J. Visual Languages and Computing*, 7, (1996) 131-174.
16. Moran, T. P.: Getting into a system: external-internal task mapping analysis, in A. Janda (ed.), *Human Factors in Computing Systems*, (1983) pp.45-49.
17. Nielsen, J.: Heuristic evaluation. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods*, New York: John Wiley (1994) 25-62.
18. Payne, S. J., Squibb, H. R., Howes, A.: The nature of device models: the yoked state space hypothesis, and some experiments with text editors. *Human-Computer Interaction*, 5. (1990) 415-444.
19. Roast, C., Khazaei, B., Siddiqi, J.: Formal comparison of program modification. In *IEEE Symposium on Visual Languages*, IEEE Computer Society (2000). 165-171.
20. Wharton, C., Rieman, J., Lewis, C., Polson, P.: The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods*. New York: John Wiley (1994) 105-140.