Journal of Trust Management

RESEARCH Open Access

Gulfs of expectation: eliciting and verifying differences in trust expectations using personas

Shamal Faily[1]*, David Power[2] and Ivan Fléchais[2]

*Correspondence:
sfaily@bournemouth.ac.uk
[1]Department of Computing & Informatics, Bournemouth University, Poole, UK
Full list of author information is available at the end of the article

Abstract
Personas are a common tool used in Human Computer Interaction to represent the needs and expectations of a system's stakeholders, but they are also grounded in large amounts of qualitative data. Our aim is to make use of this data to anticipate the differences between a user persona's expectations of a system, and the expectations held by its developers. This paper introduces the idea of *gulfs of expectation* – the gap between the expectations held by a user about a *system and its developers*, and the expectations held by a developer about the *system and its users*. By evaluating these differences in expectation against a formal representation of a system, we demonstrate how differences between the anticipated user and developer mental models of the system can be verified. We illustrate this using a case study where persona characteristics were analysed to identify divergent behaviour and potential security breaches as a result of differing trust expectations.

Keywords: Security, Usability, Personas, CSP, Trust

Introduction

When building usable software, it is accepted that both the gap between the user's goals and the means to achieve it (the gulf of execution), and the gap between software's representative state and the user's ability to perceive it (the gulf of evaluation) should be as narrow as possible [1]. However, for critical concerns such as software security, we believe that the expectations of those who use it, and those who develop it need to be aligned as well. Developers cannot directly perceive the models and metaphors users build to make sense of how their software works, and neither can users directly perceive how much trust to place in the developers' design and the software's ability to help satisfy their goals, or provide an accurate representation of its state. We focus specifically on security since it is an area of system design that requires a very clear understanding from users about the capabilities of the software and the consequences of decisions, and for which users have significant prior experiences and expectations. Consequently, we argue that there is a need to capture the differences in expectations between users and developers, but in such a way that security does not get in the way of building software, or understanding the user experiences associated with it.

To attend to user experiences, it is common to build usability design artefacts such as *personas*: narrative descriptions of fictional users that embody some user behaviour. This behaviour is identified by qualitatively analysing data collected during interviews or ethnographic research. Personas provide insights to developers about hard to reach users they might otherwise never meet [2]. Using the persona's narrative description, a designer may hypothesise a persona as suitable for an activity; such an activity might entail multiple personas either directly or indirectly collaborating with each other. However, other insights may be hidden in the persona description or related artefacts. When carefully identified and analysed, we can uncover instances where users and developers hold different expectations about how they trust the system to behave.

Because of the volume of data underpinning personas, we cannot rely on casual inspection alone to find such breaches of trust: a structured approach, supported by software tools, is preferable. Ideally, such an approach would allow us to automatically verify formal models of the personas in order to identify potential breaches of trust. While there has been interest in the use of such models to formalise human behaviours, these simplify the notion of artifacts like personas to a collection of attributes, rather than analysing actual personas [3]. Unfortunately, given that personas are grounded in qualitative data, devising formal models that model checkers can verify is difficult.

In this paper, we present the idea of the *gulf of expectation*. We define this as the gap between a user's expectations of a *system and its developers*, and the expectations held by the developer about the *system and its users*. To understand this gap, we describe how qualitative data analysis techniques and formal specification languages can be used together to evaluate the implications of differing trust expectations between a system's user community and its developers; these differing expectations can be used to identify precise differences between stakeholders' mental models of a system and its formal specification. We describe the related work upon which gulfs of expectation are grounded before presenting the main features of an approach that demonstrates this. We illustrate its use using a case study example, before discussing the implications and limitations of this work.

Related work

Personas

Personas are detailed description of imaginary people that embody shared assumptions about users of a product, data regarding users of a product, or both [4]. Unlike task models which focus on modelling activities that humans undertake, personas are concerned with their needs and expectations.

Personas were first introduced by Cooper [5] as a means for dealing with programmer biases arising from the word user. These biases lead to programmers introducing assumptions causing users to bend and stretch to meet these needs; Cooper called this phenomenon 'designing for the elastic user'; his solution was to design for a single user representing the target segment of the system or product being designed. This approach brings two benefits. First, designers only have to focus on those requirements necessary to keep the target persona happy. Second, the idiosyncratic detail associated with personas makes them communicative to a variety of stakeholders. Since their initial proposal over a decade ago, personas have now become a mainstay in User Experience practice, with articles, book-chapters, and even a book [4] devoted to the subject of developing and applying them to support interaction design.

Personas have also been useful for engaging stakeholders in quality concerns like security [2], and the analysis that contributes to their creation complements security analysis by identifying opportunities for both use and misuse [6]. A visual illustration of the role personas can play in design for security can be seen in [7].

Personas help describe the impact of human factors when imagining how a system might be used; knowing these behaviours can be useful when prioritising features or looking for innovative ideas. However, many insights that might lead to innovation may not have been obvious when originally creating personas. Such insights may be locked in the empirical data upon which the artifacts are based, and may not emerge by simply presenting them to stakeholders, or writing them into a scenario.

Deriving persona trust characteristics using grounded theory

Qualitative data analysis techniques have been shown to be effective at making sense of the data upon which personas are based. A leading approach for qualitative data analysis is Grounded Theory; this is a qualitative data analysis technique for generating theory from observed real-world phenomena [8]. These theories, and the sense-making activities associated with carrying them out, form the raw material that subsequent design activities can build upon.

Grounded Theory involves the application of three coding activities. To help make initial sense of the data, open coding is carried out to initially categorise themes arising in the data. The analytic tool of *sensitising questions* is used to help the researcher to see process, variation, and so on, and to make connections between concepts. Axial coding involves crosscutting, or relating, concepts to each other; this activity goes hand-in-hand with, rather than separately following, open coding [8]. Causes, conditions, contexts, and consequences are identified, mapped using semantic relationships, and structured into categories; this leads to new insights into the source material, and guides subsequent coding activities. During selective coding, emergent categories are organised around a central, core category according to specific criteria, such as ensuring the central category is abstract, it appears frequently in the data, it is logical and consistent with the data, and the concept grows with depth and explanatory power as other categories are related to it.

Grounded Theory is not traditionally construed as a design technique per se, but it has been used for theory-building in security and privacy research. For example, Adams [9] used Grounded Theory to induce a method of user perceptions of privacy from empirical data, and Fléchais [10] used Grounded Theory to induce and refine a model of the factors affecting the design of secure systems, based on empirical data gathered from several different case studies.

One of the barriers to exploring the efficacy of this technique is a lack of understanding between qualitative data analysis *research* concepts and usability *design* concepts. Without this understanding, it is difficult to envisage how tool support might help both design and research activities. The Persona Case framework (illustrated in Fig. 1) was designed to surmount this barrier by illustrating how qualitative models can be used to systematically derive personas [11]. This involves treating a design problem as a research problem, and devising research questions to characterise it. Qualitative research is then conducted to collect empirical data from the user population of concern; this might include running

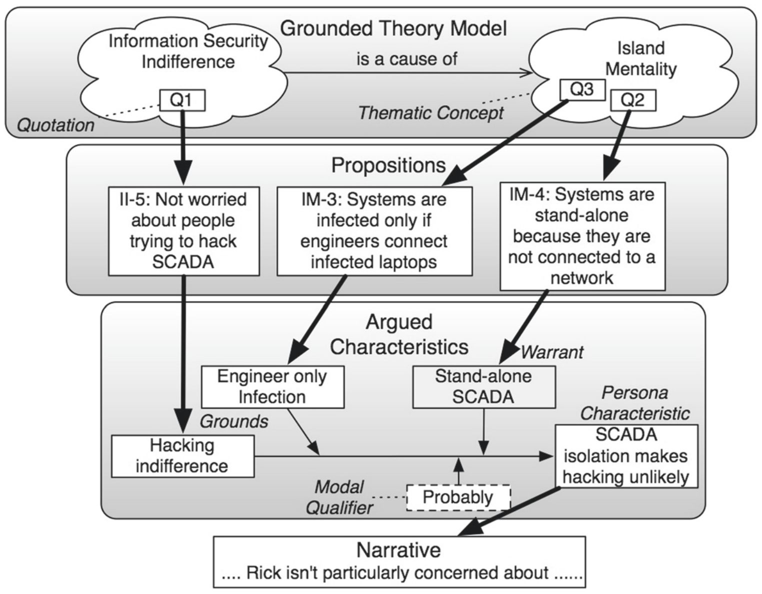

Grounded Theory Model
Information Security Indifference
is a cause of
Island Mentality
Q1
Q3
Q2
Quotation
Thematic Concept
Propositions
II-5: Not worried about people trying to hack SCADA
IM-3: Systems are infected only if engineers connect infected laptops
IM-4: Systems are stand-alone because they are not connected to a network
Argued Characteristics
Warrant
Engineer only Infection
Stand-alone SCADA
Persona Characteristic
Grounds
Hacking indifference
SCADA isolation makes hacking unlikely
Modal Qualifier
Probably
Narrative
.... Rick isn't particularly concerned about ......

Fig. 1 The contribution of Grounded Theory to personas (taken from [11])

interviews, or some form of ethnographic study. The qualitative data collected is then coded and analysed using Grounded Theory to develop a conceptual model that tackles the research problem. Using models based Toulmin's Model of Argumentation [12], each relationship from this conceptual model is structured to motivate and justify a persona's characteristic.

Recent work has illustrated how the Persona Case framework can be used to elicit *trust characteristics* from pre-existing personas [13]. These are useful for understanding the expectations about a system that personas of users expect to hold, or personas of developers expect users to satisfy. These characteristics describe the attributes of personas that lead them to behave in a trustworthy manner (intrinsic trust), and attributes of the context that motivate persona trustworthy behaviour (contextual trust). The work builds on Riegelsberger's framework for trust in technology-mediated interactions [14] to support a grounded theory analysis of trust factors influencing a particular user population. The research questions that drive the coding process are motivated by this framework, and include questions about intrinsic and contextual trust properties. The Persona Case framework is then used to derive intrinsic or contextual trust characteristics from each relationship in the grounded theory model.

Modelling and analysing users and systems with formal methods

While personas and their trust characteristics can identify the trust expectations an archetypical system user or system developer might have, it may not be apparent what the impact of expectation clashes might be in terms of system behaviour. Consequently, it may not be apparent that undesirable outcomes, ranging from unexpected system behaviour through to inconsistent use of security mechanisms, result from these clashes.

Formal methods can help identify such clashes. As the design of trustworthy systems requires input from stakeholders from different disciplines, formal methods allow contributions from these different disciplines to be represented in forms that can be understood properly and contextually by all stakeholders in the design process; it does this by providing rules about how to describe and reason about people and systems. Moreover, if models of users and the interacting system can be described precisely, claims about how easy activities are for users can be assessed, or how trustworthy a system is given their expectations. Assessing these claims is particularly important when complex interactions between system elements are difficult to make sense of, and lead to emergent behaviour that may be obvious when carrying out normal testing activities.

Dix has identified several kinds of formal methods for HCI, the most dominant classes being *user* methods that encompass models for analysing interaction, and *system* methods that model the system the user interacts with. The contribution that failures in human-automation interaction have made to aviation accidents [15], has led to innovation in a variety of different user methods for observing and analysing how users carry out tasks to achieve their goals, together with methods for modelling the outcomes of this research [16]. Although the task analysis carried out by designers to create such models can be time consuming, task analysis and modelling are still frequently used by human factors researchers, particularly those working in regulated domains like aviation and maritime. However, despite work on adapting task modelling techniques to support the *engineering* of interactive systems, e.g. [17–20], they can be problematic when considering their *design* for two reasons.

First, task analysis and task modelling were devised to evaluate human performance associated with some design in context. As such, one needs to design and implement the system before analysing and modelling how a human user interacts with it. Such systems are usually highly complex, and the role of the human is limited to those activities necessary to support them. In contrast, humans play a more instrumental role in many software systems we encounter on a day-to-day basis, such as desktop productivity tools or mobile apps; their context of use also affords greater creativity and unpredictability. While we might envisage how humans use such systems, the cost of analysing and formally modelling their use as tasks would be exorbitant.

Second, while task models of envisaged systems appear useful for modelling tasks at a high level of abstraction, they are ineffective if tasks are accomplished in different ways without significant differences in effective performance; Diaper [21] illustrates this by providing examples from Air Traffic Control where, at a high level, tasks frequently appear brief and routine, whereas at lower levels are much more elaborate.

Because of the way that behavioural nuances are manifested, modelling user interactions in system terms provide a means for identifying erroneous or suspicious behaviour on an a priori basis [22, 23]. In their review of formal verification techniques for Human-Automation Interaction, Bolton et al. [15] describe two dominant strategies for eliciting differences in modes of behaviour using *system* based formal methods.

The first strategy involves the use of theorem proving, where theories are built and proven to verify a system's correctness. This strategy was used by Masci et al. [24], who used PVS to support a field study into the social aspects of cognition by different participants. PVS (Prototype Verification System) is a framework for constructing specifications in higher-order logic that can be subject to theorem proving [25]. Masci et al. analysed different information resources and developing theories that explained physical and social work activities. While such a strategy appears to complement the activities associated with user experience research, automating this strategy can be difficult due to the expressive nature of the logics used.

The second strategy entails the use of model checking, where formal models are verified to see if they satisfy desired properties encapsulated within a specification. Bolton [20] demonstrated such an approach using Enhanced Operator Function Model with Communications (EOFMC) to model human-human protocols. The instantiated EOFMC task model was translated into SAL, and input into the SAL-SMC model checker to find countere xamples associated with potential miscommunication.

Introducing CSP and FDR

One problem associated with both of the above strategies, which arguably stymies the broader take-up of formal methods, is the creation of verifiable specifications of people and systems. In the examples described by Masci et al. and Bolton, specifications were created by analysing pre-existing system documentation and case data. However, in a real design setting, the available qualitative data may not be in such a digestible format.

While it is usually employed as a specification language for verifying concurrent protocols, CSP (Communicating Sequential Processes) has also been used for modelling patterns of interaction at higher levels of abstraction. CSP is a language for describing observable behaviour [26]. Components representing designs or properties are represented using *processes*, and processes and behaviours are described in terms of *events*. Although the process algebra used to model behaviours in CSP can be expressive, most models can be specified using only the Stop, prefix, external, and internal choice operators.

CSP is also augmented with a language of data-types, and parameters can be used to represent state information or identify instances of generic processes. If processes share parameterised events, these are expressed as *channels*. Channels include parameter values, separated by ‘ . ’, representing the data being passed.

The below specification (taken from [27]) illustrates the use of CSP to model the *Dining Philosopher's* problem [28].

```
N = 8

PHILNAMES = {0..N-1}
FORKNAMES = {0..N-1}

channel thinks, sits, eats, getsup : PHILNAMES

channel picks, putsdown : PHILNAMES.FORKNAMES

-- A philosopher thinks, sits down, picks up two forks,
-- eats, puts down forks and gets up, in an unending
-- cycle.

PHIL(i) =
 thinks.i -> sits!i -> picks!i!i ->
 picks!i!((i+1)%N) -> eats!i -> putsdown!i!((i+1)%N) ->
 putsdown!i!i -> getsup!i -> PHIL(i)
```

Behaviours can also be described using *traces*, which are sequences of events that may be performed. In large models, we are interested in sequences of events that lead to failures. Using the FDR (Failure-Divergence Refinement) model checker [29], it is possible to carry out a refinement check of one specification against the properties of another. Refinement check failures lead to traces that provide evidence of *divergences*, where undefined behaviour might arise following a particular point.

A more thorough description of CSP and FDR is beyond the scope of this paper, but a more detailed introduction to both is provided by [30].

Using CSP and qualitative data analysis to derive formal specifications

CSP is precise enough for its specifications to be formally checked, yet also expressive enough to deal with some nuances of human interactions. For example, Buth [31] used CSP, together with the FDR model checker to compare mental and system models associated with an aircraft's autopilot system, and evaluate whether the mental models were valid refinements of the system's specification. CSP has also be used to verify instance-based behavioural models such as use cases [32] and scenarios [33], and reasoning about interactions between multiple human actors; this was demonstrated by Jirotka and Luff, who used CSP to model work practices associated with shares trading [34].

If it were possible to derive and express characteristics suggesting the mental state of personas using CSP, it would be possible to use refinement checking to anticipate their likely behaviour, investigate whether this behaviour satisfies a system's safety and liveness properties, or is free from divergent behaviour. This would help to highlight patterns of behaviour that betray the trust placed by the developer (through the system design) on the user. While there is no obvious means for deriving such characteristics, Jirotka and Luff [34] indicate how this may be possible. Specifically, to elicit the CSP events and processes associated with a financial trading floor, Jirotka and Luff carried out qualitative data analysis of video transcripts; this involved summarising segments of text using categories, and drawing relationships between these categories to not only make sense of these categories, but identify hitherto invisible themes within the transcripts themselves. The categories elicited formed the basis of candidate CSP events and processes. As they began to make more sense of this data, they were able to perceive how such interactions might be formalised. Moreover, undertaking this formal analysis complemented the qualitative data analysis by providing an even better understanding of the original transcripts.

Exploiting gulfs of expectation

Our work aims to identify gulfs of expectation where a system's users have different trust expectations about a system than its developers, where both users and designers are modelled as personas. With careful analysis of descriptions of these expectations, we can elicit more precise, fine-grained models of behaviour implied by a persona. When refinement checked against a precise system specification, we demonstrate how failures verify the presence of these gulfs. This general approach is visualised in Fig. 2.

We identify gulfs of expectation by developing persona characteristics about intrinsic and contextual trust, and closely aligning the methods and tools for creating these characteristics with approaches that can formally model and evaluate them. To do this, we align the qualitative data analysis concepts associated with the Persona Case framework

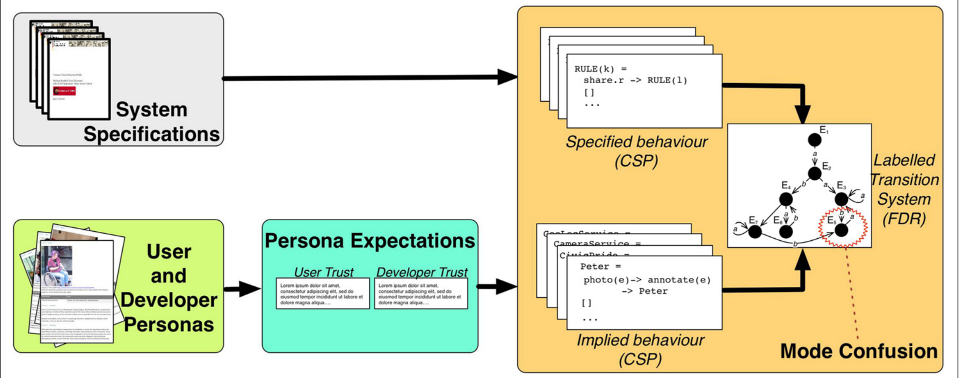

System Specifications
User and Developer Personas
Persona Expectations
User Trust
Developer Trust
RULE(k) = share.r -> RULE(l) [] ...
Specified behaviour (CSP)
Peter = photo(e)-> annotate(e) -> Peter [] ...
Implied behaviour (CSP)
Labelled Transition System (FDR)
Mode Confusion

Fig. 2 Approach for eliciting and verifying gulfs of expectation

[11] with those associated with a CSP specification, thereby creating a means of creating CSP specifications that formally describe persona behaviour.

To facilitate the required alignment of qualitative data analysis and formal modelling, we rely on tools that integrate the qualitative data analysis of persona data into the tools used to create and manage them [35]. CAIRIS (Computer Aided Integration of Requirements and Information Security) [36] is an illustration of such a tool. CAIRIS is an open-source security design tool for specifying usable and secure systems. CAIRIS was developed to better understand the form that software tools for designing usable and secure software systems might take, and was designed with extensibility in mind. CAIRIS has been used to import data from sources ranging from wiki pages and spreadsheets [37], to open source repositories about attack patterns [38]. The tool has also been validated using several real-world projects, e.g. [6, 39].

Besides providing assurance about the origin of persona characteristics, CAIRIS also allows insights that might otherwise be lost when exporting data out of Computer Aided Qualitative Data Analysis Software (CAQDAS) into tools used by software designers. To exploit gulfs of expectation, we further extend CAIRIS with additional qualitative data analysis functionality, and the ability to specify CSP processes that suggest potential gulfs of expectation.

We have also created a simple interface between CAIRIS and FDR; this facilitated the refinement checking of implied persona behaviour specifications against system specifications. Failures and divergences resulting from the model checking process verify the presence of gulfs of expectation; they highlight mismatches between persona trustworthiness, and the expectations set in a system specification about what users associated with the personas should do.

To exploit gulfs of expectation, we have extended the Persona Case framework and CAIRIS in three main areas.

Codes and code books

As indicated earlier, when undertaking a qualitative data analysis, segments of text (*quotations*) are characterised with one or more *codes*. Codes are words that capture a summative, salient, essence-capturing and/or evocative attribute for a portion of language-based or visual data [40]. The codes of particular interest to us concern *events*.

These correspond to CSP events: codes characterising interaction for processes, be these processes within a system specification, or processes implied from persona behaviour.

Codes are stored in a *code book* database, which is maintainable by team members. Based on the guidelines proposed by [41] for maintaining codes, each entry contains a full description, an inclusion criteria for when text should and should not be categorised with the code, and an example of some text categorised by the code.

Extensions for CSP

To capture the elements of gulfs of expectation, we need to associate a number of concepts with those used to specify CSP descriptions. The conceptual relationships are illustrated in the UML class diagram shown in Fig. 3.

*Implied specification*s need to be created of untrustworthy behaviour that personas might exhibit. These specifications contain descriptions of one or more *implied processes*. Implied processes are CSP process descriptions that formally specify some interaction between a persona and its environment, but this interaction is only implied by the persona. Although this behaviour is summarised narratively, it is formally specified using a CSP process description. The process description incorporates channels that are based on event codes.

The model described in Fig. 3 was implemented in CAIRIS by augmenting its database structure to include additional tables and relationships corresponding with the diagram's classes and associations respectively.

Implied processes describe behaviour motivated by a persona's *code network*. These code networks are derived from the qualitative model constructed from the grounded theory analysis upon which personas are based. However, they also incorporate the event codes that form the basis of the implied process channel. The code network usually corresponds with the conceptual model relationships underpinning a single persona characteristic. However given that, like CSP processes, personas interact with a broader environment, the network may include relationships with other codes as well.

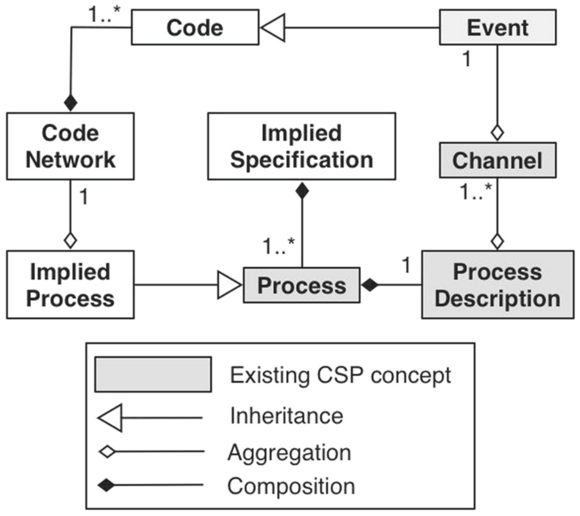


Fig. 3 Model of implied behaviour

Model checking implied specifications to verify gulfs of expectation

The implied processes can be combined into a single formal CSP implied specification for an activity corresponding to a system of interest. The combination of these processes into a single model is performed by a Java application, which interfaces to FDR.

Using FDR, we verify this generated model against the system specification to identify whether mistrusting behaviour in the implied process violates the system specification. If it does, allowable event traces that violate the system are output. These traces demonstrate failures that provide evidence of gulfs of expectation.

Case study: exploiting gulfs of expectation to identify security vulnerabilities

We now present an example of how gulfs of expectation can be elicited and exploited to identify failures leading to security problems. We consider an activity within a system's context of use, where the activity entails installing and configuring *webinos* supported software applications. *webinos* is a federated, open-source communications platform designed to support web applications running consistently and securely across mobile, PC, home media, and in-car systems [42]. We chose an example based on *webinos* as much of the design data upon which the *webinos* architecture is based is publicly available [43]; this data includes personas that motivated the *webinos* platform, and use cases specifying how users should interact with applications using *webinos*.

The software application being installed is an in-car travel game for children called “Kids in Focus”; the game is used by a user persona (Helen), and created by an application developer persona (Jimmy). Helen and Jimmy are described in more detail in [44]. The game allows Helen's son (Eric) to play an online card game with Helen's grandfather (Peter) at home. This application was designed to demonstrate how *webinos* can facilitate secure communications between an in-car telematics system and a home network. However, for this example, we will explore how the characteristics of Helen and Jimmy might lead to a security breach when installing and running this game, arising out of different trust expectations.

We begin by modelling a simple system specification for installing Kids in Focus on a tablet owned by Eric. The `SYSTEM` process uses the `pick` and `verify` channels to represent the downloading of an application and its subsequent validation. As we are only concerned with legitimate applications for this analysis, the system will always respond with `yes`.

```
datatype User = helen
datatype App = kif
datatype Device = ericsTablet
datatype Response = yes | no

channel pick : User.App.Device
channel verify : User.App.Device.Response

SYSTEM = pick?u?a?d -> verify.u.a.d.yes -> SYSTEM
```

We also refined a pre-existing *webinos* scenario called "Unsafe application install" to consider how Helen might install a *webinos* application.

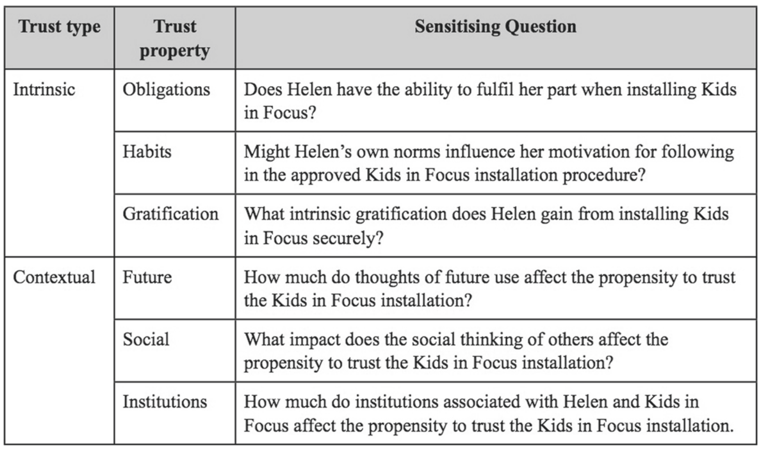
Having just chatted to some of her friends on the phone to arrange a meet-up after work, Helen realises she isn't very happy with her current diary arrangements, and the need to synchronise between her phone, tablet, and laptop.
On her way to work one morning, Helen browses the app store and discovers a webinos app that allows her to keep all of her diaries in sync automatically. Helen assumes this might simply be a cloud-based scheduler, but because she is on the train and doesn't have a huge amount of time before her train pulls in, she decides to download and setup this app on her tablet now, as it appears to be quite small.
Helen started up the app for the first time, expecting to have to install existing data from her phone's calendar. She wasn't disappointed, but first Helen was asked to walkthrough several webinos dialogues asking her to setup things called "personal devices" and "personal zones". Helen didn't know anything about what these were so, because he was on the go, clicked through every dialog box until the app started up.

Although not explicitly described in this scenario, Jimmy is an invisible collaborator by developing the functionality necessary to install the software.

Eliciting trust characteristics

We elicited trust characteristics for the pre-existing Helen and Jimmy personas by carrying out qualitative data analysis of pre-existing data; this data was collected during workshops attended by prospective users and developers. The attendees were recruited based on shared characteristics with the Helen and Jimmy personas, and the workshop considered how the participants made access control decisions. Further details about this study can be found in [45].

The workshop reports were subject to a grounded theory analysis, and Riegelsberger's framework was used to develop a series of sensitising questions to help variate and make connections between the different codes. To illustrate these, Fig. 4 shows the questions used to code Helen workshop reports.



Trust type	Trust property	Sensitising Question
Intrinsic	Obligations	Does Helen have the ability to fulfil her part when installing Kids in Focus?
	Habits	Might Helen's own norms influence her motivation for following in the approved Kids in Focus installation procedure?
	Gratification	What intrinsic gratification does Helen gain from installing Kids in Focus securely?
Contextual	Future	How much do thoughts of future use affect the propensity to trust the Kids in Focus installation?
	Social	What impact does the social thinking of others affect the propensity to trust the Kids in Focus installation?
	Institutions	How much do institutions associated with Helen and Kids in Focus affect the propensity to trust the Kids in Focus installation.

Fig. 4 Sensitising questions used to analyse reports from *Helen* workshops

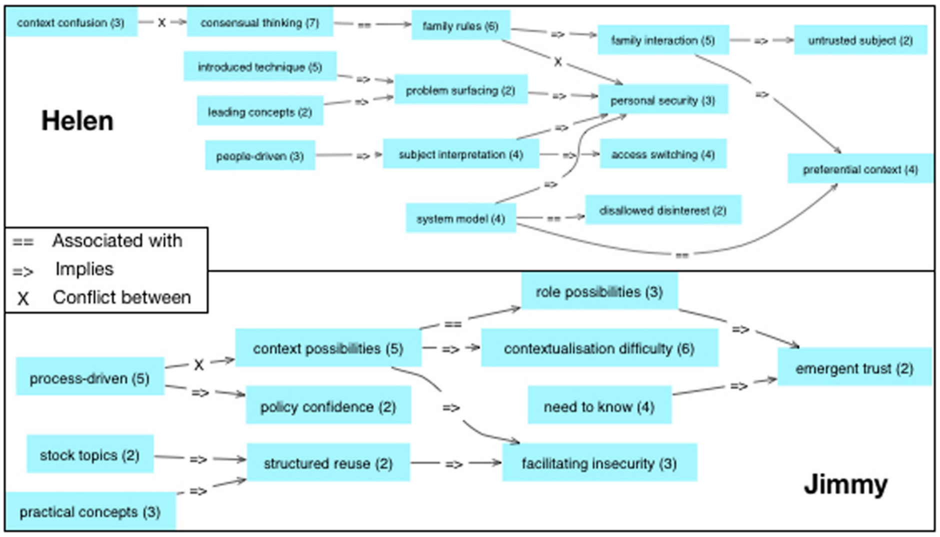
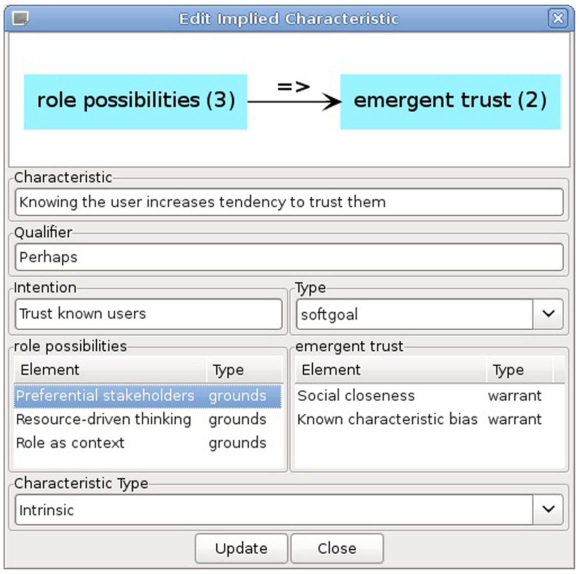


Fig. 5 Grounded theory models upon which Helen and Jimmy trust characteristics are based

From the grounded theory analysis, 57 and 37 quotations were elicited based on the Helen and Jimmy focus group reports respectively; these quotations were based on 26 codes. Figure 5 shows conceptual models generated by CAIRIS based on this analysis, and from which code networks will be based; the numbers associated with each node correspond with the number of quotations categorised with each code.

Relationship within the conceptual models form the basis of individual trust characteristic. For example, Fig. 6 illustrates how the trust characteristic *Knowing the user increases tendency to trust them* is a synopsis of the code relationship between the *role possibilities* and *emergent trust* codes.

Narrative descriptions summarising these characteristics from both personas are provided below.



Characteristic: Knowing the user increases tendency to trust them

Qualifier: Perhaps

Intention: Trust known users

Type: softgoal

role possibilities

Element	Type
Preferential stakeholders	grounds
Resource-driven thinking	grounds
Role as context	grounds

emergent trust

Element	Type
Social closeness	warrant
Known characteristic bias	warrant

Characteristic Type: Intrinsic

Fig. 6 Trust characteristic for Jimmy

- Helen Trust Characteristics
  - Intrinsic Trust: Helen's concern about her family can sometimes cloud her judgement or lead her to make access control decisions that, in the cold light of day, might not be best for her or Eric. On a good day, however, Helen has an eye for seeing intricacies in contexts, and appreciating the sort of things that strangers might want to take advantage of.
  - Contextual Trust: Thinking explicitly about how her family interacts with technology helps Helen think about how to access control data associated with these interactions. Also, because small details complicate her thought process, Helen uses things like sketches and abstractions to support the access control decision making process.
- Jimmy Trust Characteristics
  - Intrinsic Trust: Jimmy uses his standard design and development tools and processes to model and make sense of security issues. Sometimes he finds thinking about security tedious, and will re-use settings and designs where he thinks this is appropriate. This seems more efficient than thinking deeply about intricate details. Although the different ways his software might be used can be overwhelming, thinking about them gets him thinking about what some of his target users might go through when setting up and running his applications. At times, however, this appreciation can lead him to make decisions which err too much towards their freedom of action rather than the best decision given their implicit security and privacy expectations.
  - Contextual Trust: Thinking about the many different contexts can be taxing, particularly given the tools available to Jimmy are not a perfect fit for this job. Unfortunately, different contexts are not Jimmy's only worry. The problems associated with things like categorising and structuring security information, and the discussions sometimes Jimmy has to have with colleagues, means making any sort of security design decision can take forever. These considerations mean that, in the cold light of day, Jimmy is forced to make decisions that might lead to inappropriate user behaviours. This causes some angst when having to trade-off user freedom with what he considers an appropriate level of security when deciding default settings.

Eliciting candidate gulfs of expectation from trust characteristics

Once tentative trust characteristics are available, gulfs of expectation can be identified when the conventional use of a specified system is challenged. These instances constitute system mistrust as the positive estimation of trust by a trustee is either intentionally or unintentionally betrayed [46]. These instances are found by verbalising potential cases of mistrust, formalising them as CSP descriptions and, based on insights arising from this formalisation exercise, re-coding the qualitative data based on any events introduced to the shared code book.

Eliciting the trust characteristics suggested candidate implied processes for both Helen and Jimmy, as described below.

Helen

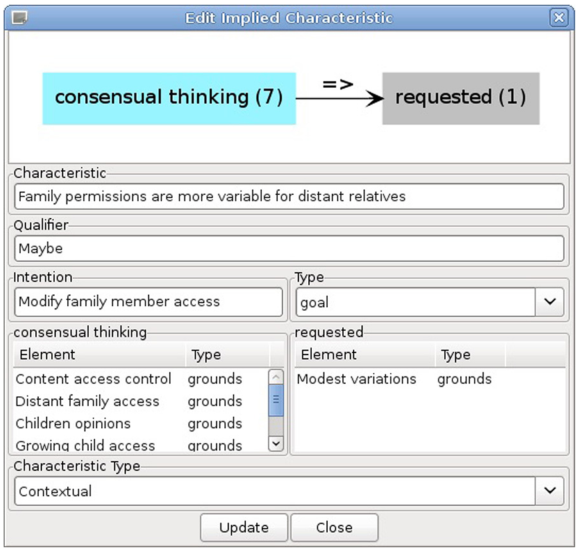
Helen appears to have a propensity for dropping or ignoring steps when making access control decisions for multiple family members at the same time. This can be contextualised by clicking through dialog boxes when responding to install wizards requesting permissions for different family members when setting up communication links for the Kids in Focus application.

When considering how such behaviour might be formalised, channels were introduced to consider how Helen might pick permissions for Kids in Focus, and what the implications for these might be. As these channels were introduced, the events associated with them were added to the codebook, and the workshop reports were re-coded with these in mind.

Eventually, this additional qualitative data analysis led to the introduction of a new trust characteristic indicating that for access controls decisions, the more distant a family member is, the more variable the result of the decision will be. The qualitative model underpinning this characteristic in CAIRIS is shown in the upper portion of Fig. 7. In this model, cyan nodes correspond with codes derived from the grounded theory analysis, and grey nodes correspond with inferred event codes.

Jimmy

The trust characteristic for Jimmy suggests that he might make spurious and unpredictable access control decisions when thinking about variations to user contexts of use. For example, thinking about the devices that Kids in Focus might run on, and where these devices are confuses Jimmy; this may lead him to specify a random default value. Jimmy does, however, believe he has a good understanding of his user community and, when deciding access control defaults, believes his users should be afforded freedom of action rather than constraints.



Characteristic: Family permissions are more variable for distant relatives

Qualifier: Maybe

Intention: Modify family member access

Type: goal

consensual thinking

Element	Type
Content access control	grounds
Distant family access	grounds
Children opinions	grounds
Growing child access	grounds

requested

Element	Type
Modest variations	grounds

Characteristic Type: Contextual

Fig. 7 Trust characteristic derived from Helen mistrust analysis

Developing implied processes from gulfs of expectation

For this example, we consider the elicitation of implied processes based on the previously described trust characteristics for Helen and Jimmy. These elicited characteristics informed thinking about ways of formalising this behaviour, leading to implied processes for HELEN and JIMMY. In these processes, we model the way that Helen goes through the process of picking an application and running it with a given set of permissions, and Jimmy assumes a user would go through the same process.

To simplify the example, the permissions have been reduced to three possible values: tooLittle, justRight, and tooMuch; these represent the permissions granted relative to those needed for the application to run successfully. Likewise, the result of running the program can have one of four possible values which represent the program crashing (crash), the program running with limited functionality limited, the program running with full functionality (full), and the program communicating private information about the user (breach).

In addition to the channels used by the SYSTEM process three additional channels are used. The requested channel represents the permissions requested by an application and the run channel represents the permissions that the user has chosen to run the application with. When the application has been run the result is shown on the res channel.

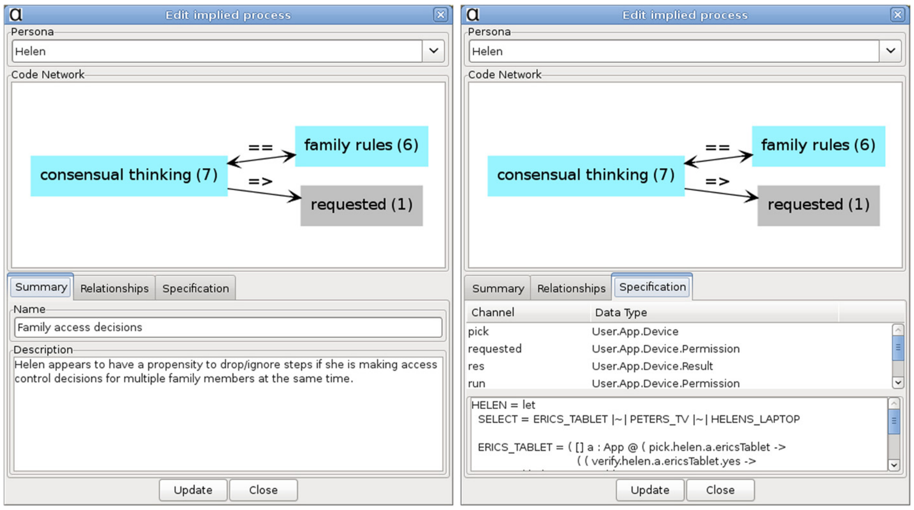
Helen implied processes

To represent Helen's propensity for dropping or ignoring steps when making access control decisions, we have modelled three separate behaviours dependent on the device that is being used.

When using Eric's tablet (ERICS_{T}ABLET), she is always cautious and tends to refuse permissions if she is too busy to think about them. Likewise, when using her own laptop (HELENS_{L}APTOP) she tends to allow whatever permissions are requested as she wants her applications to work first time. Only in the more relaxed environment of interacting with her father's television (PETERS_{T}V) does she fully consider the permissions she is setting.

```
datatype Permission =
 tooLittle | justRight | tooMuch datatype Result =
 crash | limited | full | breach datatype Response = yes | no

datatype Device =
 ericsTablet | petersTV | helensLaptop

channel requested : User.App.Device.Permission
channel run: User.App.Device.Permission
channel res : User.App.Device.Result

HELEN = let
 SELECT =
 ERICS_TABLET |~| PETERS_TV |~| HELENS_LAPTOP

 ERICS_TABLET =
 ( [] a : App @ ( pick.helen.a.ericsTablet ->
 ( ( verify.helen.a.ericsTablet.yes ->
 requested.helen.a.ericsTablet?p ->
 run.helen.a.ericsTablet.tooLittle ->
 res.helen.a.ericsTablet?r -> SELECT ) []
 ( verify.helen.a.ericsTablet.no ->
 SELECT ) ) ) )

 PETERS_TV =
 ( [] a : App @ ( pick.helen.a.petersTV ->
 ( ( verify.helen.a.petersTV.yes ->
 requested.helen.a.petersTV?p ->
 run.helen.a.petersTV.justRight ->
 res.helen.a.petersTV?r -> SELECT ) []
 ( verify.helen.a.petersTV.no ->
 SELECT ) ) ) )

 HELENS_LAPTOP =
 ( [] a : App @ ( pick.helen.a.helensLaptop ->
 ( ( verify.helen.a.helensLaptop.yes ->
 requested.helen.a.helensLaptop?p ->
 run.helen.a.helensLaptop.p ->
 res.helen.a.helensLaptop?r -> SELECT ) []
 ( verify.helen.a.helensLaptop.no ->
 SELECT ) ) ) ) within
 SELECT
```

Figure 8 shows a CAIRIS user interface for this implied process. As the UI shows, not only is the CSP specification specified, but also a short textual summary of the process, and its corresponding code network. Both the summary and the qualitative model help analysts make sense of the relationship between the model and the formal CSP specification. As this example suggests, the relationship between Helen and her family is an important facet of this implied process, so the code network extends the code relationship associated with the trust characteristic by including the 'family rules' code.



Name: Family access decisions

Description: Helen appears to have a propensity to drop/ignore steps if she is making access control decisions for multiple family members at the same time.

Channel	Data Type
pick	User.App.Device
requested	User.App.Device.Permission
res	User.App.Device.Result
run	User.App.Device.Permission

```
HELEN = let
  SELECT = ERICS_TABLET |~| PETERS_TV |~| HELENS_LAPTOP

  ERICS_TABLET = ( [] a : App @ ( pick.helen.a.ericsTablet ->
                        ( ( verify.helen.a.ericsTablet.yes ->
```

Fig. 8 Implied Process summary and specification derived from Helen's 'Family permissions are more variable for distance relatives' trust characteristic

Jimmy implied process

As for the analysis of Helen, reflecting on Jimmy's behaviour led to thinking about the considerations that might lead to access control decisions. When recoding the Jimmy transcripts with this in mind, this led to the additional trust characteristic shown in Fig. 9. To represent Jimmy's spurious decision making when confused by contextual possibilities, and his tendency to give users more freedom of action than they legitimately need, we specified the implied process description for `JIMMY` below.

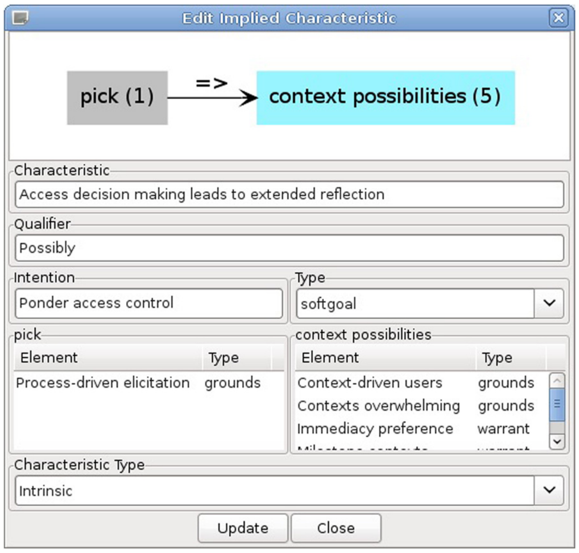
```
JIMMY = ( [] d : Device , u : User @
 ( ( pick.u.kif.d ->
 requested.u.kif.d?p -> JIMMY ) []
 ( run.u.kif.d.tooLittle ->
 res.u.kif.d.crash -> JIMMY ) []
 ( run.u.kif.d.justRight ->
 res.u.kif.d.full -> JIMMY ) []
 ( run.u.kif.d.tooMuch ->
 res.u.kif.d.breach -> JIMMY ) ) )
```

Refinement checking implied processes to verify gulfs of expectation

The `HELEN` and `JIMMY` processes were combined and imported into FDR using the CAIRIS-FDR interface, and an implied specification generated.

In this example, there is a single user process and a single application process. In a more general analysis there could be a number of each type of process. The processes `USERS` and `APPS` represent the interleaving of these two types of processes.

The `US` process is the composition of the `USERS` and `SYSTEM` process where events on the `pick` and `verify` channels are synchronised. Similarly, `UA` is the composition of the `US` and `APPS` process where events on the `pick`, `requested`, `run` and `res` channels are synchronised.



Characteristic: Access decision making leads to extended reflection

Qualifier: Possibly

Intention: Ponder access control

Type: softgoal

pick

Element	Type
Process-driven elicitation	grounds

context possibilities

Element	Type
Context-driven users	grounds
Contexts overwhelming	grounds
Immediacy preference	warrant

Characteristic Type: Intrinsic

Fig. 9 Trust characteristic derived from Jimmy mistrust analysis

```
USERS = HELEN APPS = JIMMY

US = USERS [| {| pick , verify |} |] SYSTEM UA =
 US [| {| pick , requested , run , res |}
 |] APPS
```

In the system specification, two of the possible results represent error cases, these are `crash` and `breach`. To test for these error cases, we perform two separate trace refinement checks against processes that can perform any event apart from either crashes or breaches.

```
RUN(s) = [] e : s @ e -> RUN(s) NO_CRASH =
 diff(Events, { res.u.a.d.crash |
 u <- User, a <- App , d <- Device}) NO_BREACH =
 diff(Events, { res.u.a.d.breach |
 u <- User, a <- App , d <- Device})

assert RUN({NO}_CRASH) [T= UA assert RUN(NO_BREACH) [T= UA
```

The output of the application is a number of traces that verify the presence of gulfs of expectation; these detail the sequence of events that led to the error condition.

There are two traces that result in the application crashing.

In the first of these Helen tries to install and run the Kids in Focus application on her laptop. Helen believed that the system would allow her to do this, but, due to an error in Jimmy's coding, the application fails to request all the permissions needed to successfully run the application. As Helen is likely to accept the default permissions when using her laptop, she fails to spot the missing permissions and the application crashes.

```
pick.helen.kif.helensLaptop
verify.helen.kif.helensLaptop.yes
requested.helen.kif.helensLaptop.tooLittle
run.helen.kif.helensLaptop.tooLittle
res.helen.kif.helensLaptop.crash
```

Another example of an application crashing takes place when Helen installs Kids in Focus on Eric's tablet. This time the error is not caused by Jimmy's coding, which requests the correct permissions for the application to run, but by changes in Helen's expectations between specifying permissions for the application and installing it. As Helen is very concerned about the privacy of Eric, she has denied some of the permissions needed for the application to run correctly; this has caused the application to crash.

```
pick.helen.kif.ericsTablet
verify.helen.kif.ericsTablet.yes
requested.helen.kif.ericsTablet.justRight
run.helen.kif.ericsTablet.tooLittle
res.helen.kif.ericsTablet.crash
```

The final trace results in a leaking of confidential information. This is similar to the first trace in that the application requests the wrong permissions due to a coding error. This time the application requests too many permissions, including ones that result in Helen's private data being made public. As the system's state is based on Jimmy's expectations that users should have all the freedom of action they need, this results in a breach of Helen's privacy.

```
pick.helen.kif.helensLaptop
verify.helen.kif.helensLaptop.yes
requested.helen.kif.helensLaptop.tooMuch
run.helen.kif.helensLaptop.tooMuch
res.helen.kif.helensLaptop.breach
```

Discussion and limitations

While the example presented is very constrained, we believe the approach described can be adapted to incorporate gulfs of expectation associated with more than two personas. This is particularly useful in critical systems where some level of assurance is needed about the trustworthiness of personas to undertake certain activities. For example, if we consider the activities carried out by the named traders in the dealing activities described in [34] then, assuming these traders were personas and narrative information about these personas was available, we might identify situations within which these activities may fail to be performed, or the implications of replacing one persona with another are identified which are not as expected. Moreover, when multiple designers are concurrently engaged in specifying both the system and persona-derived refinements, even trivial failures such as these may not be immediately apparent until a refinement check takes place.

While the broader concept of gulfs of expectation does not have to rely on formal methods, our approach for eliciting and exploiting gulfs of expectation relies on elements of qualitative data analysis and formal modelling in CSP. But rather than these quantitative and qualitative activities being in tension with one another, they are self-supporting. Developing a shared codebook is not incompatible with the activities for eliciting members of a CSP alphabet, and while abductive reasoning helps make sense of the causes of divergent behaviour, formally specifying this behaviour also helps make sense of the qualitative models. As a result, we believe the self-supporting nature of our approach has the potential to make both qualitative data analysis and formal specification languages more accessible to designers. Moreover, by marrying this approach with tool-support and building both qualitative data analysis and model checking capabilities into CAIRIS, divergent behaviour can be analysed at any time, rather than at fixed stages in the design process.

A limitation of our work is that no summative usability evaluation was carried out on the changes made to CAIRIS UIs to support our approach. As a result, questions remain about whether the approach might unduly influence the qualitative coding process itself. Additionally, the study described was undertaken by analysts familiar with grounded theory, persona creation, and CSP – a skills combination not readily available in many engineering contexts where this approach might be useful. However, CSP does have a relatively simple notation, and complex process descriptions can be constructed from fragments of smaller, easier to understand specifications. Moreover, by making the codes, persona-based relationships, and the relationships underpinning each CSP fragment visible, we have some means for identifying undue biases that might creep into both the qualitative and formal analysis.

The aim of this research was to explore whether it was possible to a priori explore whether persona descriptions might be analysed to uncover possible system security breaches arising from usability problems. Our research shows that this is indeed possible by exploring differences in the expectations held by different personas about the system behaviour, however more work is clearly needed to streamline this process, explore if it can be applied to concerns other than security, and apply it to more complex scenarios.

Conclusion

This paper has introduced the idea of the *gulf of expectation*. In doing so, we have demonstrated an approach for eliciting and formally verifying differing trust expectation associated with this gulf. As a corollary, we make two contributions.

First, we have shown that formal specifications that characterise different aspects of persona behaviour can be derived using the same qualitative data analysis techniques that created them. By model checking these specifications of persona behaviour with system specifications, we can conduct an a priori evaluation of the trust implications of a system. Previously, such an evaluation would have been possible only by using personas as a basis of selecting human participants when carrying out a summative evaluation of high fidelity prototypes or, as suggested by [3], as part of a more involved, layered approach for evaluating a system's assurance. Our work does not replace the need for such user testing or socio-technical design approaches, but it does illustrate how such evaluations can be carried out formatively, comparatively quickly, and during the early stages of a system's design.

Second, we developed extensions to existing software tools to provide a basis for model-checking persona behaviour. We acknowledge that the software tools described are limited, in that systematic automation is currently limited only to the categorisation of qualitative models, the specification of elicited CSP descriptions, and the combination of these descriptions into a single CSP specification of implied persona behaviour. Nonetheless, our work takes a step towards formally and economically validating usability claims that could hitherto only be evaluated by exhaustive user testing.

Acknowledgements
The research described in the paper was funded by the EPSRC EUSTACE project (R24401/GA001).

Authors' contributions
SF conceived and design the approach, designed and implemented the CAIRIS modifications necessary to support the approach, and led work on the validating case study DP contributed to the model of implied behavior, developed the CAIRIS-FDR bridge, and elicited the implied processes in the case study IF participated in the design of the approach, and helped to draft the manuscript. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Department of Computing & Informatics, Bournemouth University, Poole, UK. [2]Department of Computer Science, University of Oxford, Oxford, UK.

Received: 24 June 2015 Accepted: 21 July 2016
Published online: 29 July 2016

References
1. Norman DA (1988) The Design of Everyday Things. 1st edn. Basic Books, New York
2. Faily S, Fléchais I (2010) Barry is not the weakest link: eliciting secure system requirements with personas. In: Proceedings of the 24th BCS Interaction Specialist Group Conference. BCS '10. British Computer Society, Swindon. pp 124–132

3. Johansen C, Jøsang A (2015) Probabilistic modelling of humans in security ceremonies. In: Garcia-Alfaro J, Herrera-Joancomartí J, Lupu E, Posegga J, Aldini A, Martinelli F, Suri N (eds). Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance. Lecture Notes in Computer Science. Springer, Cham, Switzerland Vol. 8872. pp 277–292
4. Pruitt J, Adlin T (2006) The Persona Lifecycle: Keeping People in Mind Throughout Product Design. Elsevier, San Francisco
5. Cooper A (1999) The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity (2nd Edition). Pearson Higher Education, Indianapolis
6. Faily S, Fléchais I (2011) User-centered information security policy development in a post-stuxnet world. In: Proceedings of the 6th International Conference on Availability, Reliability and Security, Vienna. pp 716–721
7. Vallindras A, Faily S (2015) Designing Security through Personas. https://vimeo.com/shamalfaily/dstp. Accessed 22 July 2016
8. Corbin JM, Strauss AL (2008) Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory. 3rd edn. Sage Publications, Inc., Los Angeles
9. Adams A (2001) Users' perceptions of privacy in multimedia communications. PhD thesis. University College London
10. Fléchais I (2005) Designing secure and usable systems. PhD thesis. University College London
11. Faily S, Fléchais I (2011) Persona cases: a technique for grounding personas. In: Proceedings of the 29th International Conference on Human Factors in Computing Systems. ACM, Vancouver. pp 2267–2270
12. Toulmin S (2003) The Uses of Argument. updated edn. Cambridge University Press
13. Faily S, Fléchais I (2014) Eliciting and Visualising Trust Expectations using Persona Trust Characteristics and Goal Models. In: Proceedings of the 6th International Workshop on Social Software Engineering. SSE 2014. ACM, Hong Kong. pp 17–24
14. Riegelsberger J, Sasse MA, McCarthy JD (2005) The mechanics of trust: A framework for research and design. Int J Hum Comput Stud 62(3):381–422
15. Bolton ML, Bass EJ, Siminiceanu RI (2013) Using formal verification to evaluate human-automation interaction: A review. Syst, Man, Cybernet: Syst IEEE Trans 43(3):488–503
16. Diaper D, Stanton N (2004) The Handbook of Task Analysis for Human-computer Interaction. Lawrence Erlbaum, Mahwah
17. Palanque P, Basnyat S (2004) Task patterns for taking into account in an efficient and systematic way both standard and erroneous user behaviours. In: Johnson C. W., Palanque P. (eds). Human Error, Safety and Systems Development. IFIP International Federation for Information Processing. Springer, Boston Vol. 152. pp 109–130
18. Jenkins DP, Stanton NA, Salmon PM, Walker GH (2009) Cognitive Work Analysis: Coping with Complexity. Ashgate, Farnham
19. Paternò F, Santoro C, Spano LD (2012) Improving support for visual task modelling. In: Winckler M, Forbrig P, Bernhaupt R (eds). Human-Centered Software Engineering. Lecture Notes in Computer Science. Springer, Berlin Vol. 7623. pp 299–306
20. Bolton M, Bass E (2013) Evaluating human-human communication protocols with miscommunication generation and model checking. In: Brat G, Rungta N, Venet A (eds). NASA Formal Methods. Lecture Notes in Computer Science. Springer, Berlin Vol. 7871. pp 48–62
21. Diaper D, Stanton NA (2004) Wishing on a sTAr: The Future of Task Analysis. In: Diaper D, Stanton NA (eds). The Handbook of Task Analysis for Human-Computer Interaction. Lawrence Erlbaum Associates. Chap. 30, Mahwah. pp 603–619
22. Harrison MD, Thimbleby H (1990) Formal Methods in Human-computer Interaction, Vol. 2. Cambridge University Press, Cambridge
23. Thimbleby H (2007) Press On: Principles of Interaction Programming. MIT Press, Cambridge
24. Masci P, Curzon P, Furniss D, Blandford A (2013) Using PVS to support the analysis of distributed cognition systems. Innovations in Systems and Software Engineering 11(2):113–130
25. Owre S, Rajan S, Rushby JM, Shankar N, Srivas M (2005) PVS: Combining Specification, Proof Checking, and Model Checking. In: Computer Aided Verification. Springer, London. pp 411–414
26. Hoare CAR (1985) Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA
27. University of Oxford (2015) FDR 3 website. https://www.cs.ox.ac.uk/projects/fdr/. Accessed 22 July 2016
28. Anonymous (2016) Dining Philosopher's Problem. https://en.wikipedia.org/wiki/Dining_philosophers_problem. Accessed 22 July 2016
29. (2005) FDR 2 User Manual. Formal Systems (Europe) Limited, Oxford UK
30. Gibson-Robinson T, Armstrong P, Boulgakov A, Roscoe AW (2014) FDR3 — A Modern Refinement Checker for CSP. In: Tools and Algorithms for the Construction and Analysis of Systems, Berlin. pp 187–201
31. Buth B (2004) Analysing Mode Confusion: An Approach Using FDR2. In: Heisel M, Liggesmeyer P, Wittmann S (eds). Computer Safety, Reliability, and Security, 23rd International Conference. Springer, Berlin Vol. LNCS 3219. pp 101–114
32. Nogueira S, Sampaio A, Mota A (2008) Guided Test Generation from CSP Models. In: Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing. Springer, Berlin. pp 258–273
33. Carter J, Gardner WB (2008) Converting scenarios to CSP traces with Mise en Scene for requirements-based programming. Innov Syst Softw Eng 4(1):45–70
34. Jirotka M, Luff P (2002) Representing and modeling collaborative practices for systems development. In: Dittrich Y, Floyd C, Klischewski R (eds). Social Thinking–Software Practice. MIT Press, Cambridge
35. Faily S, Lyle J (2013) Guidelines for integrating personas into software engineering tools. In: Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. EICS '13. ACM, London. pp 69–74
36. Faily S (2016) CAIRIS web site. http://cairis.org. Accessed 22 July 2016
37. Faily S, Lyle J, Paul A, Atzeni A, Blomme D, Desruelle H, Bangalore K (2012) Requirements sensemaking using concept maps. In: Proceedings of the 4th International Conference on Human-Centered Software Engineering. Springer, Berlin. pp 217–232

38. Faily S, Lyle J, Namiluko C, Atzeni A, Cameroni C (2012) Model-driven architectural risk analysis using architectural and contextualised attack patterns. In: Proceedings of the Workshop on Model-Driven Security. ACM, Innsbruck. pp 3–136
39. Faily S (2015) Engaging stakeholders during late stage security design with assumption personas. Inform Comput Secur 23(4):435–446
40. Saldaña J (2009) The Coding Manual for Qualitative Researchers. Sage, San Diego
41. MacQueen KM, McLellan E, Kay K, Milstein B (1998) Codebook development for team-based qualitative analysis. Field Methods 10(2):31–36
42. Fuhrhop C, Lyle J, Faily S (2012) The webinos project. In: Proceedings of the 21st International Conference Companion on World Wide Web. WWW '12 Companion. ACM, Lyon. pp 259–262
43. webinos Consortium (2013) webinos design data repository. https://github.com/webinos/webinos-design-data. Accessed 22 July 2016
44. webinos Consortium (2013) webinos personas. https://github.com/webinos/webinos-design-data/tree/master/personas. Accessed 22 July 2016
45. Faily S, Lyle J, Fléchais I, Atzeni A, Cameroni C, Myrhaug H, Göker A, Kleinfeld R (2014) Authorisation in Context: Incorporating Context-Sensitivity into an Access Control Framework. In: Proceedings of the 28th British HCI Group Annual Conference on People and Computers. British Computer Society, Swindon. pp 189–194
46. Marsh S, Dibben MR (2005) Trust, untrust, distrust and mistrust – an exploration of the dark(er) side. In: Proceedings of the Third International Conference on Trust Management. iTrust'05. Berlin. pp 17–33