Formal Definitions for Design Spaces and Traces

Judy Bowen
Dept. of Computer Science
The University of Waikato
New Zealand
Email: jbowen@waikato.ac.nz

Anke Dittmar
Institute of Computer Science
University of Rostock
Germany
Email: anke.dittmar@uni-rostock.de

***Abstract*—Within the domain of interactive system development and design, particularly for safety-critical systems, there is an inherent tension between formalisms used for software engineering methodologies and the creative aspects of design. In this paper we consider how we might better unify these by way of a framework for design spaces and design artefacts. We present formal definitions for simple and complex design spaces and then describe how they are incorporated into traces. We then discuss how these can be used to reason about considerations such as preservation of requirements and iterative changes throughout the design process and provide some small examples of this.**

I. Introduction

Software engineering supports the creation of high-quality software, and it is widely assumed that the quality of a software product depends on both its inherent characteristics (or fitness for purpose) and also on the quality of the design and development process [1]. As a result numerous process models have been suggested in the literature which are used for two main purposes. Prescriptive models provide "how-to" support by telling us which activities or sub-activities have to be performed in which (however elaborated) order to get a good product. On the other hand, descriptive models help us to record and reflect upon actual design processes. It is the latter mode of use we are especially interested in in this paper.

We want to investigate how a framework for complex design spaces, such as that introduced in [2], can be used to track selective elements within interaction design processes for later analysis of performed design activities and their outcomes. A general benefit of formal process models and frameworks is that they come with well-defined concepts which allow us to understand the differences and similarities of observed design processes and to systematically capture their elements from certain perspectives. Our previous work has shown, though, that common assumptions in formal software engineering approaches need to be revisited to better embrace the richness of interaction design processes.

Central to most existing formal software development approaches is the concept of refinement. Looking through the lens of refinement, design and development processes are seen as (not necessarily linear) transformation from initial specification to final implementation. Consequently, observed design processes are typically captured in terms of evolving specifications which are expressed in some formal notation to enable formal reasoning about properties of both the product and the design process. However, successful interactive system design requires collaboration between multi-disciplinary teams, which is characterised by an elaborated division of labour and by heterogeneous design practices which become tangible in the diverse design artefacts that are used by the different sub-teams. For example, user interface sketches, prototypes, task and context descriptions, formal specifications of the user interface or the functional core of the application etc. In order to achieve quality design, different design options have to be explored and compared and choices should be informed by the viewpoints and expertise of the relevant sub-teams. Our framework of complex design spaces (described in more detail in the next section) addresses these issues by taking a broader view on how designers create and relate design artefacts while maintaining the assumption that they need to meet some initial or emerging expectations of the users (e.g., requirements or assumptions about the environment).

The central idea of this paper is to use traces in conjunction with the framework of complex design spaces to follow the evolution of an actual design process in terms of the distributed creation, modification and discarding of design artefacts and their relationships. Traces, in this context, have a complex structure and parts of them can be folded or unfolded depending on the focus of the analysis. This can be seen as an alternative to the tracking of design processes based on formal refinement which works for only some of the artefacts within the process. We further argue that the suggested tracking approach supports a 'reflection-on-action' [3] by allowing us to identify such factors as when decisions or choices are made, when alternative design ideas are discarded or whether specific requirements remain in a design process. We begin with a discussion of related work in the areas of design notations in software engineering. This is followed by an introduction to our existing framework for design spaces. We then present new formal definitions for simple and complex design spaces and their uses. We show how we can use traces to consider specific properties of these design spaces and the uses of this. Finally we give conclusions and discuss future work.

II. Related Work and Background

Design is about deliberate changes in the world to improve it [4]. Design activities are described in [5] "as the reasoning from a set of needs, requirements and intentions to a new bit of reality, consisting of a (physical) structure and an intended

use", and hence there is a need to model the future in design [4]. Diaper goes even further by stating that design requires two types of models of the world: "models of possible future worlds need to be based on models of the current world" [4]. In this section we will briefly discuss how formal software design approaches mainly focus on descriptions of possible future (software) systems and how this influences capturing approaches of actual design processes. Then, the framework of complex design spaces is presented by way of formal definitions and examples.

A. Design and Formal Software Engineering Methods

"As software engineers, we use our knowledge of computers and computing to help solve problems", Pfleeger [1] states in her textbook. Most software development approaches including formal methods are strongly rooted in the rational problem solving paradigm. Design is about understanding the problem and then solving it This view is to be found in the first stepwise refinement methods [6] which consider software development as a sequence of small transformations (steps) from abstract specifications of the software system to be built to more concrete specifications by ensuring correctness. But it is also to be found in more recent formal approaches.

For example, García-Duque et al. [7] reject the idea of pure incremental refinement because "some design decisions may have to be reconsidered after they have been taken". The authors consider instead three basic types of specification evolution: refinement, abstraction, and retrenchment. While refinement adds knowledge to the current specification by preserving all knowledge from previous stages, abstraction results in discarding part of current knowledge or augmenting the set of allowed behaviours. Two specifications are retrenchments if they have a common abstraction. In other words, retrenchment allows us to revise a specification in a controlled way which preserves knowledge from previous stages. It also supports the exploration of different possible solutions by going back (abstraction) and forth (refinement). García-Duque et al. understand design processes as processes of knowledge construction. However, they apply their approach only at the level of requirements specifications. The explicit tracking of the relationships between the evolving specifications supports the stakeholders' understanding of what has been accepted, refined, or discarded.

Bosse et al. [8] use traces within a temporal logic approach to capture dynamic properties of a design process. A trace describes the history of a design process as a sequence of design states (S1,S2) over time, with S1 being a requirements state (problem description) and S2 being a design object description state (solution specification). The example properties that are given in [8] reveal much of the underlying assumptions on design processes. It is assumed, for example, that requirements 'stabilise' and become fixed at some point of time. While Bosse et al's work is aimed at the design of re-usable components rather than interactive systems (and therefore a different set of design criteria) there are common properties to our framework. For example the revision of design objects and co-ordination of different processes within a design task. However their solution specification assumes a full problem specification that must be satisfied rather than the co-evolution of problem and solutions that we consider. Moreover the elements within their specification space are all of corresponding types and can therefore be considered using refinement relations, which is not the case in our heterogeneous design spaces.

Our work is influenced by Morgan's [9] uniform approach to refinement in which he does not distinguish between requirements, specifications and implementations but calls everything a program. A program is seen as a contract between a client and a programmer which has to be negotiated. Refinement in this context concerns the maintenance of utility: the client gets at least what they had before or even better. Although Morgan stays within the problem solving paradigm for design, by assuming a hierarchical decomposition and refinement of programs (resulting in programs that are executable on a computer), he emphasises that stakeholders often act in both roles - the role of client and programmer. As such, in our framework we consider all stakeholders to be involved in complex networks of user-designer relationships. So that, for example, a designer in one part of the process becomes the user for some other design team in a separate part of the process.

B. Interactive System Design and the Framework of Complex Design Spaces

Interaction design processes put more emphasis on the intended use of the system to be designed. This not only concerns the usefulness or usability of the interactive software system itself but also real-world consequences [4]. Interaction design, in a broader sense, is about shaping interaction spaces between multiple people and devices in a complex web of interactions [10]. As mentioned in the introduction, people with diverse viewpoints and expertise are needed and this may be one reason why interaction design research and practices developed from many influences, including ideas from design rationale [11] and from the reflective practice paradigm [3]. We discuss the background of our framework of complex design spaces in more detail in [2], [12]. Here, those aspects of the framework are briefly presented which are essential for the suggested definitions and use of traces.

The central concepts of our framework are design spaces populated by design artefacts, and the relationships that exist between those artefacts. The design artefacts themselves are any (and all) artefacts used within the design process, irrespective of their nature. So, for example, a formal specification, a task analysis model or a sketched prototype of a dialog are all considered at the same level of abstraction and importance as an artefact. It is this heterogeneity of artefacts (which reflects the heterogeneity of the design process and design teams) that motivates us to look beyond traditional refinement methods which is restricted to specific artefacts only.

There is a need then to consider the ability of designers to relate these different artefacts in a cohesive manner, which

ensures that requirements and expectations (both those of the end-user and those of the other design teams) are adhered to. This can be used to compare different design options and make decisions/choices resulting in the discarding of some artefacts and the refinement of others (where here we use refinement in its broadest sense). A design space is the conceptual space of a design team with a 'contract' with users and with a specific 'design culture'. This 'design culture' or 'distinct design practice' concerns the type of design goals, which design artefacts are in use and types of relationships (e.g., ensure safety properties as a typical design goal, formal specifications as artefacts, refinement as a means to relate them in a 'formal method-culture'). A design space is the conceptual gathering together of all design artefacts (and the relationships between them) that are provided, created, modified, discarded by the team in order to fulfill the contract.

A design space has an entry point and an exit point, with the artefacts at those points representing the contract. In some sense we can consider them the pre- and post-conditions of the space. Design artefacts provided by the users via the entry point express their expectations and assumptions; designers deliver their resulting artefacts via the exit point to the users. Provided artefacts can represent multiple viewpoints which may contradict each other (we discuss this further below). The design spaces have a recursive structure. They can consist of sub-spaces assigned to sub-teams, describing the division of labour. Members of sub-groups are typically not disjoint and the complex design space describes the network of user-designer relationships people are involved in as well as the total set of artefacts under consideration.

Within our framework we also consider the distinction between design *alternatives* and *variants*. These describe local choices between design options within a sub-space and delayed choices by providing options to other sub-teams to include their viewpoints and expertise. Such distributed decision making is important, as every new idea (or refinement) that occurs within a design sub-space can introduce side effects which need to be considered at a more global level.

III. Design Spaces and Traces

The above description of design spaces suggests a static view, where all spaces can be considered at a single point in time. The reality, however, is that the process is inherently temporal, with sub-spaces existing for only parts of the process and dynamic changes over time. Our initial formal definition for static design spaces was given in [2], but here we expand these definitions within our trace theory which enables us to consider different user groups and multiple viewpoints as well as temporal properties. Using traces in this way allows for further reasoning about the design process as it occurs throughout the design spaces.

Our goal is to follow the evolution of a design process in order to identify such factors as: when decisions or choices are made; when alternatives are discarded; when variants are generated; when decisions are finalised; whether specific requirements remain.

A. Traces for Simple Design Spaces

Our trace concept is based on states of design spaces. Let $\mathcal{D}$ be a global set of design artefacts created and used in the context of a design project, $\mathcal{R}$ a global set of relationship types between design artefacts from $\mathcal{D}$ and $\mathcal{S}$ the set of involved stakeholders. We first consider simple spaces and define their states as follows.

Definition 1 (State of a simple design space): A *simple design space state DS* is a 5-tuple $(DT, Us, D_{entry}, D_{exit}, DRel)$, where

- $DT \subseteq \mathcal{S}$ is the design team.
- $Us = \{U_1, U_2, ...\}$ with $U_i \subseteq \mathcal{S}$ $(i = 1, 2...)$ is a finite set of user groups.
- $D_{entry} \subseteq Us \times \mathcal{D}$ with $(U, d) \in D_{entry}$ if user group U provides design artefact d to the design team DT.
  $Entry_{DS} = ran\ D_{entry}$ is the set of design artefacts in the *entry point* representing external design constraints and requirements from multiple viewpoints.
- $D_{exit} \subseteq \mathcal{D} \times Us$ with $(d, U) \in D_{exit}$ if design team DT provides design artefact d to the user group U.
  $Exit_{DS} = dom\ D_{exit}$ is the set of design artefacts in the *exit point* representing the design outcome.
- $DRel \subseteq \mathcal{R} \times Entry_{DS} \times Exit_{DS}$ with $(r, d_1, d_2) \in DRel$ if there is a relationship of type r between design artefacts d_1 and d_2 from the entry and exit point.

♢

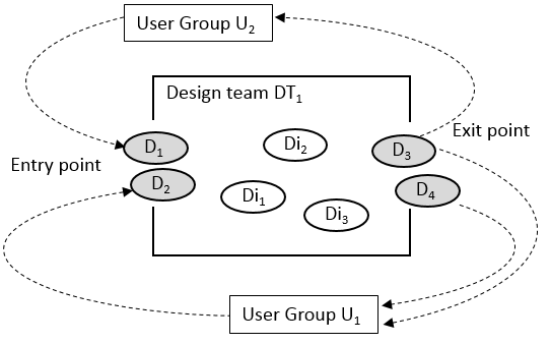

Fig. 1. A simple design space.

Figure 1 depicts an abstract simple design space. At this stage, two user groups U_1, U_2 have expressed their expectations on the design team DT_1 by means of design artefacts D_1, D_2 and the team has provided D_3, D_4 as outcome. The figure indicates that the design team may have created artefacts within the design space $(Di_1, Di_2...)$ to interpret the multiple views of the user groups and to develop and compare design ideas. Those internal artefacts and their relationships (not shown in figure 1 for the sake of clarity) are hidden to the users and they are not considered in the state definition which views a simple design space as a 'black box'. We refer to such artefacts as *local*. However, the relations between design artefacts from the entry and exit point (*DRel* in the above definition) are also based on internal relations and how they are propagated. Such propagation is straightforward in a formal

refinement approach. If in figure 1, for example, $D_1 \sqsubseteq Di_2$ and $Di_2 \sqsubseteq D_3$, then $D_1 \sqsubseteq D_3 \in DRel$ (assuming that $\sqsubseteq$ is transitive). Design artefacts that exist in both D_{entry} and D_{exit} are considered *global* as they are visible in a wider context.

Some refinement relationships between design artefacts are discussed in our previous work and mentioned here in brief only. Refinement can be based on a) formal methods (to build the system in the right way), b) lightweight notions (for a transition between informal and formal designs), c) validation techniques (to build the right system), and d) on reflection (to ensure that the design process is right). In this paper, the existence of certain relationship types between design artefacts (set $\mathcal{R}$ above) and corresponding propagation mechanisms are assumed, but not further elaborated here. A propagation mechanism p in this context is a mapping $p : Seq\ \mathcal{R} \rightarrow \mathcal{R}$.

Now, a trace of a simple design space (in short, *simple trace*) is a time-indexed sequence $\langle t_1 : DS_1, t_2 : DS_2, ...\rangle$ of states $DS_1, DS_2, ...$ of this space with $\langle t_1, t_2, ...\rangle$ a linearly ordered sequence of points of time ($t_1 < t_2 ...$). Such traces allow us to capture the dynamics of design spaces by identifying what happens at entry and exit points over time, taking into account both iterations within a design space and also the fact that different users may provide additional artefacts at different points in time. It allows us explicitly identify when design artefacts are replaced by others, created or discarded, and also when there are changes in the user groups over time. We discuss such reasoning about design spaces shortly.

B. Traces for Complex Design Spaces

A design space is called complex if it consists of sub-spaces which are design spaces themselves. Figure 2 illustrates a complex design space with three simple sub-spaces assigned to sub-teams $DT_{11}, DT_{12}, DT_{13}$. In complex spaces, new designer-user relationships typically develop between the design sub-teams. In figure 2 sub-teams DT_{12}, DT_{13} are users of design outcomes of sub-team DT_{11} and this results in new requirements and constraints for DT_{11} (indicated by design artefacts Di_3, Di_4 as un-shaded ellipses).

We now introduce different views on states of complex design spaces to be able to consider traces of design processes at different levels of granularity. In particular, we distinguish between the unfolded form and the folded form of a state, shortly referred to as unfolded/folded state. An *unfolded state* DS of a complex design space consisting of n sub-spaces is defined as tuple $(DS_1, DS_2, ..., DS_n)$ with DS_i being the states of the sub-spaces ($i = 1, 2, ..., n$). Due to the recursive nature of complex spaces, their states are described by nested tuples of unfolded sub-states until the level of simple design space states (defined above). Figure 2 illustrates this 'unfolded view'.

In contrast, figure 3 depicts the complex design space of figure 2 as a black-box hiding sub-spaces and internal designer-user relationships and corresponding artefacts. Here, the complex space is viewed as a simple space and this is reflected in the following definition of *folded states*. Similarly to definition 1, sets $\mathcal{D}$, $\mathcal{R}$, and $\mathcal{S}$ are given. Furthermore, let $\mathcal{P}_{\mathcal{R}}$ be a set of propagation mechanisms over $\mathcal{R}$ as introduced in the previous sub-section.

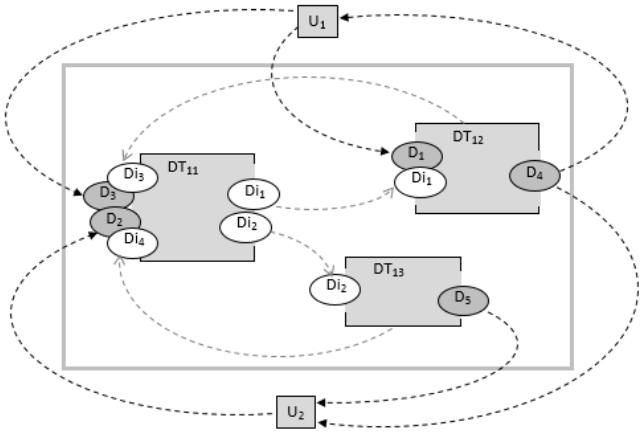

Fig. 2. A complex design space with three sub-spaces.

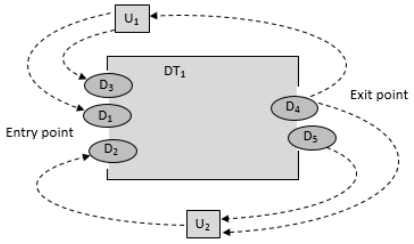

Fig. 3. Complex design space of figure 2 as blackbox.

Definition 2 (Folded state of a complex design space): Let $DS = (DS_1, DS_2, ..., DS_n)$ be the unfolded state of a complex design space. $DS^{Fold} = (DT, Us, D_{entry}, D_{exit}, DRel)$ is the *folded state* of DS if the following conditions are hold.

- $DS_i^{Fold} = (DT_i, Us_i, R_{entry_i}, R_{exit_i}, DRel_i)$ are the folded states of the unfolded sub-states DS_i ($i = 1, 2, .., n$).
- $DT \subseteq \mathcal{S}$ with $DT = \bigcup_{i=1..n} DT_i$.
- $Us = \{U \mid \exists\, i = 1..n \bullet U \in Us_i,\ \nexists j = 1..n \bullet U = DT_j\}$.
- $D_{entry} = \{(U, d) \mid \exists\, i = 1..n \bullet (U, d) \in D_{entry_i},\ U \in Us\}$ and $Entry_{DS} = ran\ D_{entry}$.
- $D_{exit} = \{(d, U) \mid \exists\, i = 1..n \bullet (d, U) \in D_{exit_i},\ U \in Us\}$ and $Exit_{DS} = dom\ D_{exit}$.
- $DRel \subseteq \mathcal{R} \times Entry_{DS} \times Exit_{DS}$ with $(r, d_1, d_n) \in DRel$ if $\exists (r_1, d_1, d_2), (r_2, d_2, d_3), ..., (r_{n-1}, d_{n-1}, d_n) \in \bigcup_{i=1..n} DRel_i$ and $\exists p \in \mathcal{P}_{\mathcal{R}}$ with $p(\langle r_1, r_2, .., r_{n-1}\rangle) = r$.

$\diamondsuit$

Similar to simple design spaces, the dynamics of complex design spaces is tracked by time-indexed sequences of space states (based on linearly ordered sequences of points of time $\langle t_1, t_2, ...\rangle$). But in contrast to simple spaces, complex spaces can be captured at different levels of granularity. A first distinction can be made between unfolded and folded traces.

- An *unfolded trace tr* is a sequence $\langle t_1 : DS_1, t_2 : DS_2, ...\rangle$ of unfolded states $DS_1, DS_2, ...$ of a complex design space.

- A *folded trace* tr^{Fold} is a sequence $\langle t_1 : DS_1^{Fold}, t_2 : DS_2^{Fold}, ...\rangle$ of folded states $DS_1^{Fold}, DS_2^{Fold}, ...$ of a complex design space.

However, the definition of unfolded and folded states enables a whole array of state descriptions. For example, sub-states can be unfolded to a specified depth in the recursive structure of sub-spaces (denoted as DS^n, with n being the number of unfolded levels), or alternatively, single sub-states only could be unfolded/folded. Correspondingly, traces of complex spaces can contain an amalgamation of both folded and unfolded state descriptions of sub-spaces and thus detail only those parts of a design process which are of particular interest in the analysis.

C. Reasoning about Design Spaces

We now describe, in general terms, how we might use the definitions above to elicit information about the design process such as where/when decisions have occurred that affect the global state space of design artefacts. We might consider these as the points where certain aspects of the design are finalised, where choices are made or where specific requirements are introduced, removed or resolved. We illustrate the description with some specific examples.

1) Reasoning about Simple Space States: For simple design spaces we use a state description, which hides internal decision making. However, where the design artefacts at the entry and exit points are related (see *DRel* in the definition) we can draw conclusions. The following examples concern the persistence of design artefacts and underlying requirements, constraints or design ideas.

- Let $d_1 \in Entry_{DS}$. If there is no tuple $(r, d_1, d_x) \in DRel$, where $d_x \in Exit_{DS}$ then we can consider that d_1 has been discarded or ignored. In some cases d_1 may not be considered or in any way changed (depending on the type of artefact, is it a design constraint or prototype etc.) but rather just leave the design space in its original state. In such cases we will have what we refer to as the 'identity relation', given by the tuple $(r, d_1, d_1) \in DRel$ which makes this explicit.
- Conversely, if for $d_1 \in Entry_{DS}$ there is a tuple $(r, d_1, d_x) \in DRel$, where $d_x \in Exit_{DS}$ then the artefact d_1 has been changed into, or used to inform, a new design artefact d_x, and has therefore been in some sense, preserved. The exact nature of this preservation depends on the nature of the artefacts and their relationship r. Design specifications may be refined, but d_1 could also be a design constraint or requirement which is preserved in a design product d_x as part of its design rationale.

2) Reasoning about Complex Space States: For complex design spaces our folded states allow us to treat design spaces in the same ways as for simple spaces (at the same level of detail). However the unfolded states (where different levels of unfolding are possible, as discussed) allow for some reasoning about internal decision making and how things are finalised across states of sub-spaces. In the following (abstract) examples, let DSS_1 and DSS_2 be the states of two sub-spaces which form part of a complex design space.

- If there are design artefacts d_1, d_2, d_3 such that $(r, d_1, d_2) \in DRel_1$ and $(r, d_2, d_3) \in DRel_2$ (r not being the identity relation) then we can consider d_1 as having been refined or utilised in DSS_1, but not finalised. That is d_1 is preserved (in its new form d_2) to form part of further work within DSS_2.
- Let $d_1 \in Entry_{DSS_1}$, $d_{x_1}..d_{x_n} \in Exit_{DSS_1}$ and $d_{x_1}..d_{x_n} \in Entry_{DSS_2}$ where $(r, d_1, d_{x_i}) \in DRel_1$ $(i = 1..n)$ and all are discarded in DSS_2 with the exception of d_{x_j} $(j \in \{1, .., n\})$. Then we can consider d_1 as having been refined or utilised in a delayed decision process across DSS_1 and DSS_2.

In previous work we have introduced the concepts 'alternative' and 'variant' to discuss distributed decision making among different options as indicated in the second example above. Alternatives refer to local processes of generating ideas and decision making which are hidden by simple spaces and their states, variants require global decision making and are visible in unfolded states of complex spaces. We do not discuss this further here, a more thorough discussion is beyond the scope of this paper.

3) Reasoning about Traces: Traces reveal something about the nature of a specific design process and the progress over time. From simple traces, and likewise from folded traces of complex design spaces we can, for example, draw conclusions about how the view on requirements and design constraints has changed during a design process. This is illustrated by the following (abstract) example where $\langle t_1 : DS_1, t_2 : DS_2, t_3 : DS_3\rangle$ is a trace capturing three iterations in a simple design space of design team *DT* as follows.

DS₁ :$Us = \{U_1, U_2\}$, $D_{entry} = \{(U_1, d_1), (U_2, d_2), (U_2, d_3)\}$, $D_{exit} = \{(d_1', U_1), (d_1', U_2)\}$, $DRel = \{(r, d_1, d_1'), (r, d_2, d_1'), (id, d_3, d_3)\}$

DS₂ :$Us = \{U_1, U_2\}$, $D_{entry} = \{(U_1, d_1), (U_1, d_4), (U_2, d_2), (U_2, d_3)\}$, $D_{exit} = \{(d_2', U_1), (d_2', U_2)\}$, $DRel = \{(r, d_1, d_2'), (r, d_4, d_2'), (id, d_3, d_3)\}$

DS₃ :$Us = \{U_1\}$, $D_{entry} = \{(U_1, d_1), (U_1, d_5)\}, ...$

For simplicity, imagine d_1, d_2, d_3 to be requirements imposed by two user groups U_1 and U_2. After the first iteration (represented by trace$\langle t1 : DS_1\rangle$), d_1 and d_2 are preserved in the design outcome d_1' provided to both users. The state DS_2 captured at time t_2 reveals that in the second iteration an additional requirement d_4 by user U_1 emerged. While d_4 is preserved in the revised design outcome d_2' the needs of U_2 are now ignored (d_2) or still not considered (d_3). The state DS_3 captured at time t_3 shows that user U_2 has withdrawn from the design process. Furthermore, U_1 replaced d_4 by requirement d_5. While simple traces allow us to identify when design artefacts have been replaced by others there is no knowledge about the relationships between these old and new design artefacts.

Unfolded (or partly unfolded) traces of complex design spaces support reasoning about distributed design activities, occurring sequentially or in parallel, with local and global iterations within and across sub-spaces. Traces allow us to detect inconsistencies between sub-spaces in their use of de-

sign artefacts, for example, whether/when revisions of design artefacts are lost again in a complex space.

As another example of an inconsistent situation, consider three sub-spaces assigned to teams DT_1, DT_2 and DT_3. Let us assume that, at time t_1, DT_1 has provided design outcome d_1' which preserves d_1 and d_2 and team DT_2 has provided outcome d_2' which preserves d_1 and d_3. Both outcomes are part of the entry point of sub-team DT_3. A design iteration is performed in sub-space DT_1, initiated by the replacement of d_1 by d_4 in the entry point. The revised outcome d_4' replaces d_1' in the entry point of sub-space DT_3. However, d_1 is still preserved in d_2' provided to DT_3 by DT_2.
The detection of such inconsistencies can not only inform the design of the interactive system under consideration but also a redesign of the design process itself (leading to a reconfiguration of sub-spaces).

IV. Discussion

Interactive systems are typically designed with an iterative approach, acknowledging that quality design is rarely achievable without iteration of design solutions and without an active involvement of users and other stakeholders. Reasoning about traces of design spaces in the manner shown provides a method of eliciting properties of the design based on the changing nature of those artefacts, i.e. it reflects this iteration. It also highlights the attention that is paid to the interests of individual user groups and stakeholders.

Some of the limitations of our proposed approach arise from the detail within (particularly complex) spaces and the recursive depths that may occur. However, our examinations of mapping actual design processes to the framework [13] (in limited form only) do not suggest that this will be the case. That is, while the number of design spaces may be large, the recursive nesting within complex spaces is fairly shallow. Further investigations are required to satisfy ourselves that this is likely to be the majority case.

While the overhead of creating and using the traces is fairly low, in that it provides a lightweight abstraction at the highest levels, the details of the underlying relations and artefacts is in reality far more detailed. However, many of the relationships are typically already visible within the engineering processes (by way of refinements of specifications and the design history for prototypes etc.) so that we might rely on capturing these existing details rather than having to create everything from scratch.

The benefits are the ability to provide both high and low level views of the emergence, introduction, discarding and finalising of various artefacts within the process. This enables us to track particular design properties (particularly useful in safety-critical systems) and ensure that requirements that may not be traceable via the formal specification and refinement processes directly can still be tracked. In addition we can better reflect on collaboration within complex design processes and detect potential weak points. For example, we can identify where some sub-spaces are not provided with the necessary design artefacts or where a redesign of sub-spaces would be useful (a refinement of the design process itself).

V. Conclusion and Future Work

In this paper we have presented formal definitions for simple and complex design spaces as well as traces which can be used to consider the evolution that occurs within a design process. We have also given examples to show how these might be used to consider properties (such as persistence and finalisation) of design artefacts that may be important when reflecting upon the process as a whole. This is particularly pertinent for the design of safety-critical interactive systems where traceability within the design process (especially with respect to safety requirements) may form part of the engineering process requirements.

The work described here forms part of ongoing research, and as such presents the detail of just one aspect of this work which is currently under investigation. Our next steps are to incorporate these findings with previous work to consider a real-world example from the domain of safety-critical interactive systems to demonstrate the applicability of our work in this area as well as the ability to identify known problems from existing case-studies.

References

[1] S. L. Pfleeger, *Software Engineering: Theory and Practice*, 2nd ed. Prentice Hall, 2001.

[2] J. Bowen and A. Dittmar, "A semi-formal framework for describing interaction design spaces," in *8th ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 2016, pp. 229–238.

[3] D. Schön, *The Reflective Practioner: How Professionsal Think in Action*. Basic Books, 1983.

[4] D. Diaper, in *The Handbook of Task Analysis for Human-Computer Interaction*, D. Diaper and N. Stanton, Eds. L. Erlbaum Associates Inc., 2004, ch. Understanding Task Analysis for Human-Computer Interaction.

[5] K. Dorst, "On the Problem of Design Problems - problem solving and design expertise," *J. of Design Research*, vol. 4, no. 2, 2004.

[6] N. Wirth, "Program Development by Stepwise Refinement," *Commun. ACM*, vol. 14, no. 4, pp. 221–227, 1971.

[7] J. García-Duque, J. J. Pazos-Arias, M. López-Nores, Y. Blanco-Fernández, A. Fernández-Vilas, R. P. Díaz-Redondo, M. Ramos-Cabrer, and A. Gil-Solla, "Methodologies to evolve formal specifications through refinement and retrenchment in an analysis-revision cycle," *Requirements Engineering*, vol. 14, no. 3, pp. 129–153, 2009.

[8] T. Bosse, C. M. Jonker, and J. Treur, "Formal analysis of design process dynamics," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 24, no. 3, pp. 397–423, Aug. 2010.

[9] C. Morgan, *Programming from Specifications (2nd ed.)*. Prentice Hall International (UK) Ltd., 1998.

[10] T. Winograd, in *Beyond Calculation*, P. J. Denning and R. M. Metcalfe, Eds. New York, NY, USA: Copernicus, 1997, ch. The Design of Interaction, pp. 149–161.

[11] H. Rittel and M. Webber, "Dilemmas in a General Theory of Planning," *Policy Sciences*, vol. 4, pp. 155–169, 1973.

[12] J. Bowen and A. Dittmar, "Coping with design complexity: A conceptual framework for design alternatives and variants," in *Interact2017*. Springer, 2017.

[13] ——, "Identifying the interplay of design artifacts and decisions in practice: A case study," in *Interact2017*. Springer, 2017.