A Domain Specific Language for Contextual Design

Balbir S. Barn and Tony Clark[1]

Middlesex University, Hendon, London, UK, NW4 4BT

Abstract. This paper examines the role of user-centered design (UCD) approaches to design and implementation of a mobile social software application to support student social workers in their work place. The experience of using a variant of UCD is outlined. The principles and expected norms of UCD raised a number of key lessons. It is proposed that these problems and lessons are a result of the inadequacy of precision of modeling the outcomes of UCD, which prevents model driven approaches to method integration between UCD approaches. Given this, it is proposed that the Contextual Design method is a good candidate for enhancing with model driven principles. A subset of the Work model focussing on Cultural and Flow models are described using a domain specific language and supporting tool built using the MetaEdit+ platform.

1 Introduction

This paper examines the role of user-centered design (UCD) to the design and implementation of a mobile social software application for supporting student social workers in their work place. The principles and expected norms of UCD raise a number of issues which lead us to propose that these problems are a result of the inadequacy of precision of modeling the outcomes of UCD, which prevents model driven approaches to method integration between UCD and established software engineering practice. A particular UCD approach - Contextual Design [3] is explored in detail from a model/language design perspective by first critiquing the key issues of ambiguity and lack of precision of diagrams normally produced as a result of Contextual Design activities. Following on from this, a subset of Contextual Design, namely, the Cultural Model is developed in terms of abstract and concrete syntax together with its accompanying semantics diagram using an approach to language design described by Clark et al [7]. An implementation using the MetaEdit+ tool [17] is also briefly described. The issues of a lack of precision of UCD methods represents an ongoing research challenge in the field of requirements engineering and a key outcome of this paper is to encourage discussion of these problems and lessons to enable method re-engineering of UCD practice.

The paper contributes to current research in human centred software engineering by providing a formal syntax and semantics for aspects of the Contextual Design methodology and in doing so provides a route whereby the exploration of how UCD and software engineering can be integrated.

The remainder of the paper is structured as follows: section 2 introduces key aspects of UCD and model driven development; section 3 presents the key motivation of this work, our experience of using UCD and the lessons learnt, and provides a short introduction to Contextual Design, the UCD approach we have selected to be subject to a formal treatment; section 4 puts the case for using a *model driven* approach to UCD; section 5 presents the approach we have taken to develop a modelling language with more precise syntax and semantics; section 6 presents the domain specific language version for Contextual Design along with an illustrative example of its use; section 7 presents concluding remarks and notes for further research.

2 Background to UCD and MDD

2.1 From User-Centered Design to Participatory Design

A detailed review of the literature concerning user-centered design is not possible within the constraints of this paper but it is useful to present an overview of key phases in development of user engagement in systems design processes. User-Centered Design (UCD) or the variant, User-Centered Systems Design [22] emerged in the 1980s as an important development recognizing the move from batch computing to interactive computing applications where there was a need to involve users in the design process. At that time, however, as Marti and Bannon [20] indicate: UCD did not imply that "users were . . . active participants in the design process", rather they were studied, observed, measured as a way of gathering requirements for the system development [8]. An implication of UCD is thus one of where the designer (hopefully) reacts to feedback from the user. A more radical school of user involvement is that attributed to the so-called Scandinavian Model of UCD, namely, Participatory Design (PD) that emerged from the research activity of people such as Bjerknes et al [5]. In PD, users are seen as equal partners in the design and development of systems. This involvement of users implies users as "active agents" and later became known as Cooperative Design [11] or more latterly as "Co-Design". In PD, interestingly, there is a focus on primary work processes and identification of technology to enhance and better support work activities. (The basis of business process modeling). As UCD concepts became established they were further elaborated as ISO Standard 13407, Human-centred design processes for interactive systems [23]. These concepts were developed and extended into 12 key principles for UCD by Gulliksen [15].

While the mantra of involving users in the design process is now well ingrained [26] it has been contested and more recently Marti and Bannon [20] outline caveats where they argue that involving users can present problems. The characterization of problems they have identified forms part of the evaluation of our experience of co-design when applied to our development of a mobile application for e-learning, and which led us to consider how such issues may be addressed by model driven practice for UCD.

2.2 Model Driven Development

Orthogonal but related to UCD is the need to recognize that software engineering development methods have also evolved and more recently model driven development is increasingly seen as critical to good design: see [10] for an overview of MDD where it is argued that modelling is a key technology that is necessary to address the representation gap between human understanding of complex modern systems and their implementations and where precision at all levels of development is key to the increased scope of computer-based support for systems development. We argue that precision is key to increasing all aspects of system quality including reliability, usability, efficiency, and that MDD offers an approach that provides precision from a range of appropriate perspectives. MDD is increasingly being used for user-centred aspects of systems such as HCI [27,25,24] and safety [1].

Modeling in general is viewed as a capstone of many software engineering approaches where it is used to as an approach to user requirements definition and as a basis for developing information systems to meet those requirements. Models provide a vehicle for explaining and sharing understanding of complex problems and provide capabilities for different views of the underlying problem at different levels of abstraction. Model driven architecture takes this premise further by providing an overarching conceptual structure for using and applying transformations to models in a structured and controlled manner in all stages of the software engineering development process.

The Object Management Group (OMG) provides a set of standards to express models and model-model transformation and has been leading industry initiatives in the promotion of technologies, methods and standards under the banner of model driven architecture (MDA) [13]. Our position is: MDA has key role to play in systems development and are in agreement with Constantine and Lockwood [8], who assert that UCD can be ambiguous and vague. In contrast, Gulliksen et al assert that "model driven approaches represent a move away from user-centered design reducing their involvement to that of the users being informants rather than co-designers". This assertion needs re-visiting in the light of MDA approaches to user interface design and recent advances in domain specific languages. Certainly Fisher [9] has identified that Collaborative Design and meta-design (using MDA principles) are key themes facing software engineering research and practice.

3 Experience of User Centered Design

This section describes some of our experiences from a recent research project that utilised UCD and software engineering approaches to developing a mobile application for Social Work education. We discuss some of the key issues and lessons arising from that experience and present an argument for model driven UCD.

3.1 Case Study

The motivation for exploring how model driven principles could be applied to user centred design arose from a recent research project where we applied a variant of user centred design to design and implement a mobile device application to support Social Work Education in the UK.

In common with many other professions, the training of social workers requires students to be placed in social work settings and to undergo assessment in the workplace. Trainee social workers in England (those on an accredited social work degree (UG or PG) must successfully complete 200 days in a practice setting. Such placement can occur in different size blocks according to structures and requirements of individual degree programmes. These requirements are maintained and regulated by the Social Work professional body – the General Social Care Council (http://www.gscc.org.uk/Home/).

During the practice learning process, there are several key stakeholders involved, including: the student; the practice mentor and assessor; the University academic tutor; the work based supervisor. The key outcome of the placement is a report that outlines the skills and competencies raised along with supporting evidence collected from the placement.

Given these background concepts the research project aimed to develop a set of applications both mobile and web-based that supported student social workers in the planning and design of practice learning assessments and in the collation of research and practice evidence towards a final report.

3.2 Experience using UCD

The project team assigned to the project was multidisciplinary. There were academic experts from Computer Science, Sociology, Social Work, along with practitioners from the Social Work field. Further, the project development teams were located in multiple locations across the UK South. As well as the multiple disciplines located within the team, the Computer Science team further represented alternative approaches to systems design, with representation from both MDA and UCD. These different approaches led to some creative tension manifested in early debates similar to that discussed between Gulliksen and Constantine. Given the make-up of the project team, it was essential to agree to a methodology that could accommodate disparate views. The team had previous experience of using a co-design process for developing mobile applications for the Nursing domain [21]. Hence this approach was adapted to suit the needs of this project and the software engineering principles influencing members of the team. Thus the project deployed a variety of methodological techniques that draw upon software engineering, social sciences research and usability.

3.3 Problems Encountered

While the system was successfully developed, its deployment and use was very limited, and is consequently still ongoing (past the project completion date).

This is partially attributed to the implementation of the co-design approach and it is here that it is considered that there are many lessons to be learnt. Using the putative framework of problems identified by Marti and Bannon [20] as a starting point the following lessons are presented:

user types The intended software applications were designed for several types of users.

users as designers While it is accepted that all users can design at some level – that is have ideas, think creatively about different uses of tools and convey those thoughts in some form explicit knowledge transfer – it is clearly not the case that users have the necessary design skills to engage in all stages of the design process.

new technologies Mobile technology is evolving at a rapid pace. Increasing power, capability and software applications possible makes it very difficult for non-technologists to remain abreast with such change. In order for users to make a significant contribution to the design process they need to have a logical understanding of technological solutions in order to be able to conceptualize new scenarios of use. This problem manifested itself very early in the co-design process: many of our participants had their first direct contact with current mobile devices in our show and tell workshops.

work environments Project champions tend to be located at management level where there is often limited understanding of operational requirements. This can have a detrimental effect on active user involvement throughout the design process. The Social Work environment in a public sector setting meant there were work pressures that often prevented users from securing sufficient time to effect a meaningful engagement in the co-design activities.

deployment risk The need for sufficient training, guidance, support and impact assessment in the work environment also need to be sufficiently defined. Issues of risk and technology in the workplace, although correctly identified at the beginning of the project manifested themselves resulting low usages in the work place context. There has been considerable interest within the social sciences in developing ideas related to risk. Beck in his seminal text Risk Society [2] argued that the "technisisation" of risk derives form the omnipresence of technology. The experience here could be seen as reflecting fear on a number of dimensions. Reluctance to engage with the project could be seen as a fear of technology itself and the ability of some individuals to cope with technological demands. The social work task is in itself high risk and high profile and the use of technological devices for training purposes could be seen as representing a 'reflexive' form of risk. The findings could also reflect the so-called 'precautionary principle'. Practitioners and students were anticipating possible difficulties in areas such as confidentiality and data protection which prevented them from considering the possible full benefit of the opportunity offered by the project.

user confusion Confusing what users want with what they truly need. Numerous user studies and approaches can create a wide and detailed user understanding however such studies can create confusion with what users say they want with what they really need.

multi-faceted design team The make-up of the design team can influence the nature of user involvement. For example, a team that is equipped with skills in UI, prototyping, and software design will likely involve users at stages in the design process. A team with predominantly HCI researchers will likely involve more users and at more stages. In this case, there was a relatively balanced team in terms of skills and knowledge – our problems were arriving at shared common vocabularies, and attempting to involve all users and all design team members all the time.

4 The Case for Modelling in User Centered Design

These lessons or observations from the co-design approach have the potential to be mitigated by taking a model-driven integrated approach to the artifact development from the co-design activities. This paper argues that artifacts from user-centred design should be model based so that transformations between viewpoints can be integrated. This requires a user-centered design approach that is both rich, for capturing key user requirements and is also model driven such that it can be subject to model driven transformations during the design and implementation process.

Currently, UCD approaches are strong on user engagement and communication but tend not to be model-based in the software engineering sense. Thus it is difficult to derive a single viewpoint to meet both the needs of stakeholders and software engineers. Such design-slicing could be a powerful feature in presenting key features of an overall design without information overload.

Multiple viewpoints are a recognized approach to such a challenge but tend to driven by software engineering needs. For example, Rational Unified Process [18] has attempted to integrate user centered design activities. Such models serve software engineering well but present notational and technique challenges to the stakeholder in the usability domain. Here, it is proposed that multiple viewpoints that are driven from UCD method approaches have the potential to reduce or mitigate the problems/issues raised earlier. Hence it is proposed that taking steps to move UCD to a more model driven software engineering approach has the potential to be more effective than taking steps to make Software Engineering more UCD focussed.

4.1 Contextual Design

The Contextual Design approach described by Beyer and Holtzblatt [3] is a good candidate for enhancing using model driven principles as it already exhibits language that one might see comfortably in the software design arena but is still a rich user-centered design approach. The method supports the production of a number of artifacts such as: key customer data as the basis for decision making; processes where work is done; interactions using "flow models"; cultural models for capturing intuitive elements of environment; consolidation using affinity diagrams.

These elements are present in a number of models enumerated here and we also indicate if there is an existing language and notation feature available from the Unified Modeling Language (UML) [14] available:

Artifact Model: produces the key customer data relevant to the system. These data are referred to in other models and also inform the technical architecture element that is part of the User Environment Design stage. The Conceptual Design method does not advocate data modeling approaches explicitly but the strong similarity suggests that UML Class Models would be a strong technique for capturing such data.

Flow Model: is an analysis tool that is used to capture communication and coordination between roles. It will typically describe what interactions take place such as request for information, supply of information or an action. In Contextual Design such a model is informal – but it can easily be modeled in UML in a variety of ways. For example flow models have been described by Activity Diagrams with additional features in [**?**].

Cultural Model: is an analysis tool that shows the cultural or political forces in the organization. Issues addressed are forces that may impinge on roles to prevent or modify how work is done. For the purposes of this paper, we view cultural forces having an effect on belief values held by individuals who are participating in a given task.

Sequence Model: shows the detailed steps that are performed to accomplish a task. The terminology used in Contextual Design shows strong correspondence to process modeling and can be represented by UML Activity Diagrams. This correlation provides a useful language tool for sharing of information between usability designers and systems designers. For the purposes of this paper we view sequence models as capturing the many alternative workflows that individuals can undertake when directed to perform a given task. The choice between different workflows is partially determined by the belief values held by a given individual. It follows that cultural models have a part to play in influencing workflow choices.

Contextual Design provides a stage that aims to consolidate findings from the analysis. In this stage tools such as Affinity Diagrams for mapping issues across the organization, and Consolidated work models for identifying common strategies are available. These tools help in addressing the problems of multiple types of users, users as designers, confusing what users want and what they truly need because an overall view is possible. Such models could be captured using stereotypes and UML class diagrams.

Similarly, process models described from different user perspectives may be organized in models so that commonality and variability may be explicitly specified. However, no formal language for expressing such variation is described. Interestingly, the method also has elements that are focused on software architecture These elements produce artifacts that include object models and the functions and structures needed by the re-designed systems expressed as a detailed architectural model.

4.2 The Semantics of Cultural Models

We now look at the Cultural model in detail and in particular examine the concepts represented in Cultural models from the perspectives of key components of language design - abstract and concrete syntax and the accompanying semantics. Inspection of examples of cultural models [4]) raises key questions. At first the diagrams appear to convey a significant amount of information. A closer examination requires a full answer to many questions. Many of the diagrams represent Influencers as overlapping circles of different sizes. Consider the following: Is there a significance in the size of circles? Is there significance to the overlaps? What does an overlap mean? Are the length of the arrows important and do they signify anything? Is there a particular style to annotating the circles and arrows? Even if the questions can be answered - and it make take many pages of explanatory text there are still issues of interpretation between users and stakeholders. Generally, these are the same class of problems that modelling design methods have addressed and which led to the standardisation of the Unified Modelling Language (UML) [14] and solutions to these problems boil down to the creation (and agreement) of an abstract syntax (a set of concepts, relationships and well-formed rules), a concrete syntax (typically a graphical notation that supports the concrete syntax), and semantic model for the domain which will allow unambiguous interpretation of instances of concepts from abstract syntax. A model-enhanced version of this diagram would therefore need to be based on a language that is specifically designed to represent cultural forces in an organization - the domain. Such a class of language is often termed a domain specific language (DSL) [6].

5 Approach

Our approach to the problem of formalising Cultural model aspects of Contextual Design (CD) is based on the principles of model driven language engineering [7]and through a process of analysis of the problem space, a domain specific language (DSL) is created. The second step is to implement the language using a meta modeling tool or by hand-coding.

5.1 Model Driven Language Engineering

A language definition must be provided using a suitable meta-language that can represent the key features of a language (shown in figure 1):

concrete syntax is the human-friendly representation of a language. The concrete syntax defines how the language is to be presented on the screen or the page.

abstract syntax is a machine friendly representation of language. The abstract syntax defines the information structures that are used to represent the essential features of the language so that they can be processed as data values by a machine without worrying about how they are displayed on the screen or the page.

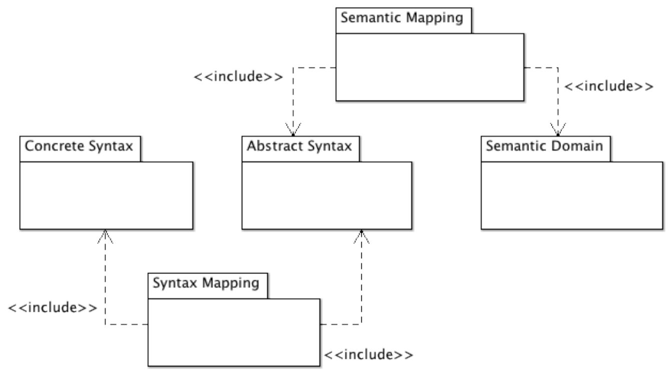


Fig. 1. Model Driven Language Engineering

syntax mapping relates the abstract syntax structures to their valid concrete representation. A syntax mapping is used by a tool to link what the user sees to how the language is stored internally.

semantic domain is a definition of the things we mean when constructing models in our language. The relationship between syntax and semantics is the same as that between relational database schemas and the tables that conform to them: there are conformance rules and there may be many databases that conform the same schema.

semantic mapping are the conformance rules between the syntax and the semantics. For example, the semantic mapping defines the rules by which a database table is considered to be correct with respect to a given schema.

In addition to the elements defined above, a language definition must contain well-formedness rules that define when concrete syntax, abstract syntax and semantic domain elements are valid. These correspond to database rules that, for example, require all column names to be unique.

Simple UML-style class models and associated constraints can be used as a suitable meta-language for representing the language components listed above. The syntax and semantics are represented as independent class diagrams and the mappings are class diagrams that include elements from the appropriate models and define relationships (associations) between the elements. There are other meta-languages available e.g. MOF [12]. Also, Halpin and Morgan's work [16] was used as the meta-modelling language for Archimate - the enterprise architecture modelling language [19].

5.2 Tooling

The next step is to provide an implementation of the language, that is, a tool that provides a binding of the various syntactic and semantic models and thus allows users to construct cultural models of the target (or subject) domain. To develop a proof of concept of this toolset, we utilised the meta modelling toolset

MetaEdit+ [17]. Meta Edit+ is a software toolset that supports the design and implementation of domain specific languages. It uses a meta modelling language GOPRR that is broadly similar to one that we used to specify our abstract syntax and has the following concepts: **Graph** specifies one modelling language such as Cultural Model; **Object** describes the basic concepts of a modeling language. Objects are the main elements of the language; Examples include the concept of Force, Role and so on; **Relationship** describes the properties for the objects' connections such as inheritance, call and transition; within the toolset, the relationship mechanism is used to form bindings with objects and roles; **Role** specifies the lines and endpoints of relationships; **Property** defines the attributes which can be used to characterise any of the previous concepts. Properties are of different data types and can be used to link to external concepts. The abstract syntax for Contextual Design was encoded in the GOPRR modelling language within the MetaEdit+ toolset in order to define the concrete syntax and the production of the accompanying tool.

6 A DSL for Contextual Design

Contextual Design (CD) invoves four different types of model: flow models; sequence models; artifact models; contextual models. It is important that we understand what these models mean in order to use them effectively. This section applies the approach to language design described in section 5.1 to CD.

6.1 Abstract Syntax

The abstract syntax is the cornerstone of a language definition. In principle there may be many different concrete syntax models and many different semantics for the same abstract syntax model. This section defines the abstract syntax for each of the main models in the CD modelling language.

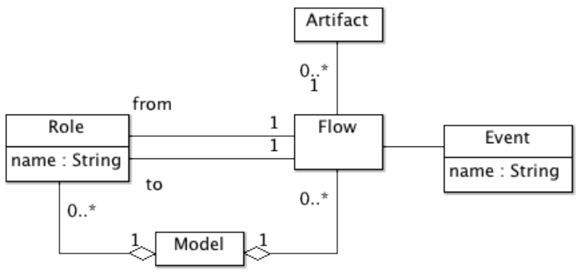


Fig. 2. Flow Models

Flow Models Figure 2 shows the abstract syntax of flow-models. The element *Model* is used as the top level container for all CD model elements. A model consists of a collection of roles with flows between them. Each flow represents an interaction between roles and is labelled with both an event that it generates and the artifacts that are involved in the interaction. An example of a well-formedness rule associated with the class *Model* is: *every role must have a unique name.*

Artifact Models Artifact models are equivalent to class models in UML. They describe the elements that are involved in the interactions between roles. In terms of our language definition, we do not need to consider artifacts in any more detail.

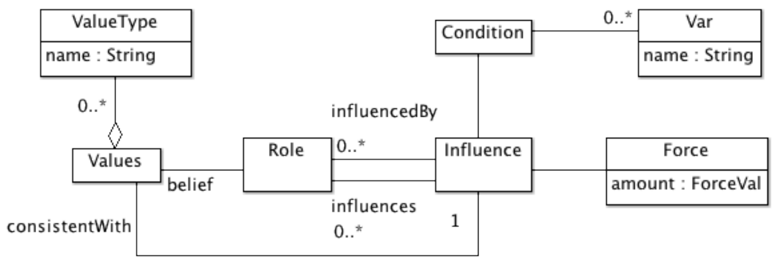


Fig. 3. Cultural Models

Cultural Models Figure 3 shows the abstract syntax of cultural models that represent influences by one role on another. Each influence has a force associated with it from *weak* to *strong*. Each role manages a collection of values that represent personal beliefs. For example an individual might believe that certain types of technology are effective or that the cost associated with using certain processes is very high. In a CD model, *Values* is an assembly of belief value types in the same way that a class has an assembly of attributes. As we shall see, the specific belief values associated with a person who performs the role is defined in the semantics. An *Influence* together with its *Force* defines a condition which must be met by any valid instance of the belief values associated with an any influenced *Role*. A condition is a boolean expression in terms of variables. The effect of applying an influence to a role is to restrict the set of possible belief values that the role can have. A well-formedness rule that applies is *Influence*: *the set of variable names in the condition must be a sub-set of the value type names associated with the belief values of all influenced roles.*

Sequence Models Figure 4 shows the abstract syntax of sequence models. Each role has an interface of activities. Each activity provides a description of what to do when an event is received by the role. Each activity has a number

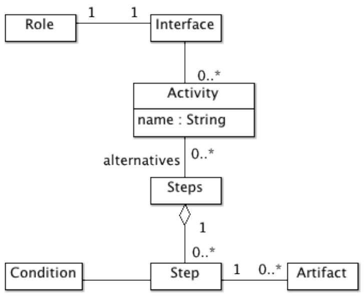


Fig. 4. Sequence Models

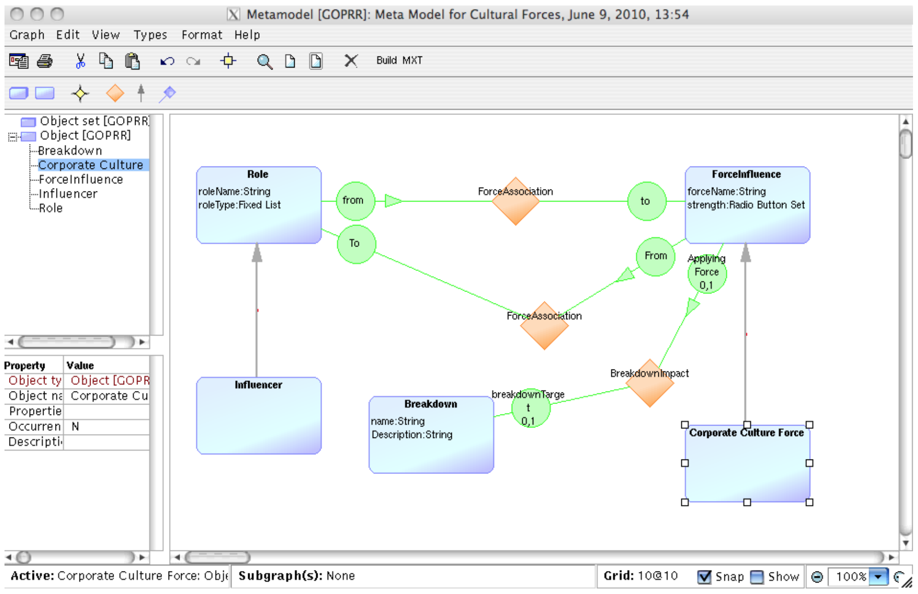
of alternative step assemblies (*Steps*) that reflects the options an individual has when responding to a request to perform a task. For example, if an individual is requested to implement a software component they may choose to implement the component in one of a number of programming languages and using any one of a number of development methods. Each individual step in a step-assembly processes some artifacts and must satisfy a collection of belief-values. The idea is that a step cannot be performed by a role unless it is consistent with the beliefs of the particular individual.

An example of a well-formeness rule for sequence models is: *the artifacts associated with a step must be a subset of the artifacts associated with the flow that gave rise to the event.*

6.2 Concrete Syntax

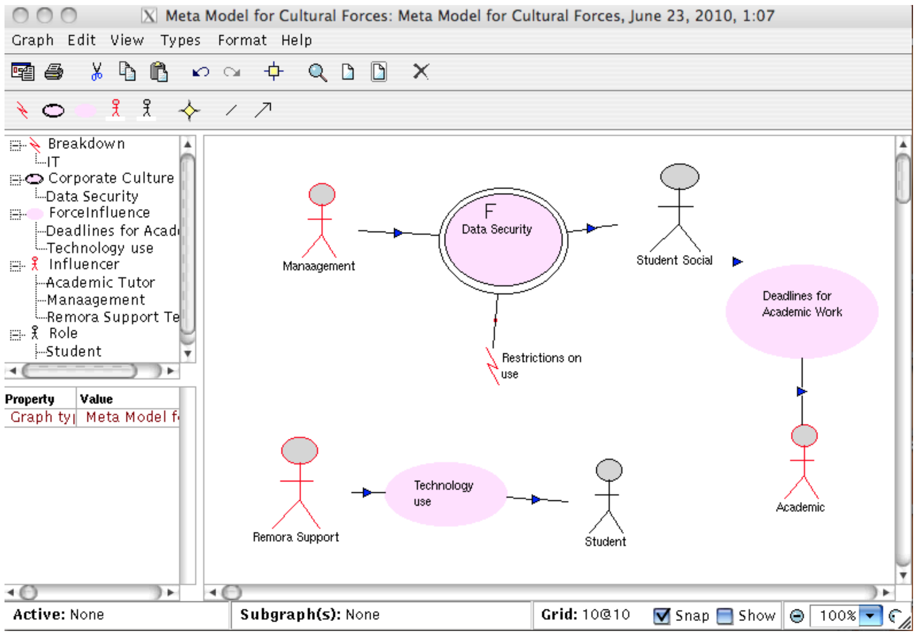
The complete abstract syntax for Contextual Design is large. In this paper we have focussed on the Cultural Forces Model as it presents concepts that address areas of the systems design process that are often not captured in software engineering. Consequently we have translated that section of the abstract syntax to the GOPRR meta modelling syntax for Meta-Edit+. The tool capability allows the creation of an concrete syntax - the notations and graphical elements and their binding to the GOPRR representation of the abstract syntax. Diagram 5 shows the abstract syntax for cultural models in MetaEdit+ and figure 6 shows the resulting cultural modelling tool generated from the meta-model and also a partially drawn cultural model of the Social work Domain.

In this diagram example Roles include Student Social Worker. Influencers, those who can assert a force and therefore influence how an activity is performed include: Management, and Academic. Examples of Forces that are brought to bear on a role include the fear of Data security (the loss of data) that was identified during the project deployment. When that force became sufficiently critical, there was a Breakdown (red lightning icon) which resulted in an restriction of the use of the mobile devices. The Toolset for developing and presenting Cultural



Property	Value
Object ty	Object [GOPR
Object na	Corporate Cu
Propertie	
Occurren	N
Descripti	

Fig. 5. GOPRR Meta Model



Property	Value
Graph ty	Meta Model f

Fig. 6. Toolset for Modelling Cultural Forces

Models was generated from the set of abstract concepts described in GOPRR, the concrete syntax developed using the graphics tools available in MetaEdit+ and the bindings and rules for how connections work were declaratively specified again in MetaEdit+

6.3 Semantic Domain and Semantic Mapping

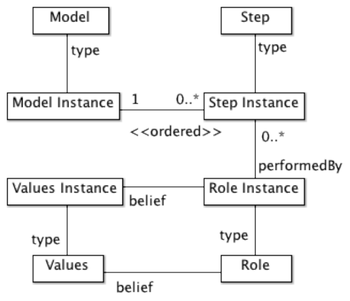


Fig. 7. Semantic Domain

Figure 7 shows both the elements of the semantic domain (classes suffixed with *Instance*) and elements of the syntax domain together with semantic mapping associations between them (labelled *type*). The semantic domain defines the elements that we are denoting using the syntax models. In this case the semantic elements are essentially sequences of step instances that have arisen from the steps associated with activities in the sequence model. However we cannot associate *any* sequence of steps with a model instance because we must satisfy the semantic mapping constraints that are outlined below:

1. *in every role instance the belief values must satisfy the condition on every influencer of the associated role.*
2. *in every step instance, the condition must be satisfied by the belief values associated with the corresponding role instance.*
3. *a step can only be associated with a role instance where the corresponding role has an incoming event with the same name as the activity giving rise to the steps.*

7 Concluding Remarks

The motivating work - the development of a mobile application for a complex domain (Social Work) highlighted that there are potential problems that arise with using co-design and while the core principles of UCD are clearly desirable, the nature of the artifacts that are produced do not transfer to the software engineering community in a straight forward manner. Thus our experience also confirms that there is still mileage on the need to converge “on a science of design through the synthesis of design methodologies” [9]. In particular there is interest are in how design theories, user centred design approaches in general and their outputs can be modeled such that method integration with established software engineering approaches can be more formalized . Hence there a role for model driven engineering in user centered design and this paper has outlined how one established UCD approach may be adapted to make it more model driven (and so artifacts captured using UML modeling tools).

CD models, as defined in the literature, have an informally defined semantics. This limits what can be achieved, especially in terms of tooling to support CD. This paper has taken a precise meta-modelling approach to the definition of a language for CD modelling. In doing so, we have defined both the syntax and (a) semantics for CD. Our semantics defines CD models to denote chains of steps that arise from interactions between roles in a business context and which process business artifacts. The semantics reflects the choices that occur in a business environment that are resolved in terms of belief-systems of the individuals involved; it does this by allowing a single model to denote multiple possible sequences of steps for each single business activity. The semantics attributes influencing factors to the ability of individuals in a business to affect belief systems and thereby influence the way that influenced individuals implement given tasks.

As a result of taking a semantics driven approach to our CD modelling language we can now perform analysis of models. For example, it is possible to determine whether, given a set of influencers on individuals, there are any sequences of steps for a given business interaction. Suppose that this is used in a business that encourages new employees to get advice from established employees when performing tasks. Our semantics allows us to determine whether there are particular sets of ’old-hands’ whose collective advice would be unhelpful. Furthermore, we can measure the amount of positive influence that mentoring is likely to have in terms of the reduction in confusion when staff take on a new role. For researchers, future projects will likely consider and evaluate further how such approaches may be used to allow more alignment with the software engineering model driven architecture paradigm.

References

1. Ra Basnyat, Nick Chozos, and Chris Johnson. Incident and accident investigation techniques to inform model based design of safety critical interactive systems. In *Design, Specification and Verification of Interactive Systems 2005*, pages 51–66. Springer Verlag, 2006.
2. U. Beck. *Risk society: towards a new modernity.* Sage Publications Ltd, 1992.
3. H. Beyer and K. Holtzblatt. *Contextual design: defining customer-centered systems.* Morgan Kaufmann Pub, 1998.
4. H. Beyer and K. Holtzblatt. Contextual design. *interactions*, 6(1):32–42, 1999.
5. G. Bjerknes, P. Ehn, M. Kyng, and K. Nygaard. *Computers and democracy: A Scandinavian challenge.* Gower Pub Co, 1987.

6. T. Clark, A. Evans, and S. Kent. Engineering modelling languages: A precise metamodelling approach. *Fundamental Approaches to Software Engineering*, pages 242–260, 2002.
7. T. Clark, P. Sammut, and J.S. Willans. Applied metamodelling: a foundation for language driven development. 2008.
8. L.L. Constantine and L.A.D. Lockwood. Usage-centered engineering for Web applications. *IEEE software*, pages 42–50, 2002.
9. G. Fischer. Software engineering themes for the future. In *Proceedings of the 28th international conference on Software engineering*, page 1044. ACM, 2006.
10. Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.
11. J.M. Greenbaum and M. Kyng. *Design at work: Cooperative design of computer systems*. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1991.
12. Object Management Group. OMG Meta Object Facility, 2010. `http://www.omg.org/mof/`.
13. Object Management Group. OMG model driven architecture, 2010. `http://www.omg.org/mda/`.
14. Object Management Group. Unified Modeling Language, 2010. http://www.uml.org/.
15. J. Gulliksen, B. Goransson, I. Boivie, S. Blomkvist, J. Persson, and Å. Cajander. Key principles for user-centred systems design. *Behaviour & Information Technology*, 22(6):397–409, 2003.
16. T. Halpin and T. Morgan. *Information modeling and relational databases: from conceptual analysis to logical design*. Morgan Kaufmann, 2008.
17. MetaCase Inc. Metaedit+ workbench - build your own domain-specific modeling language, 2009. `http://www.metacase.com/mwb/`.
18. P. Kruchten. *The rational unified process: an introduction*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.
19. M. M Lankhorst, H.A H.A. Proper, and J Jonkers. The Anatomy of the ArchiMate Language.
20. P. Marti and L.J. Bannon. Exploring User-Centred Design in Practice: Some Caveats. *Knowledge, Technology & Policy*, 22(1):7–15, 2009.
21. D. Millard, Y. Howard, L. Gilbert, and G. Wills. Co-design and Co-deployment Methodologies for Innovative m-Learning Systems. *Multiplatform E-Learning Systems and Technologies: Mobile Devices for Ubiquitous ICT-Based Education*, 2009.
22. D.A. Norman. Cognitive engineering. *User centered system design*, pages 31–61, 1986.
23. International Standards Organization. *Human-centered design processes for interactive systems*. ISO, 1999.
24. Frank Radeke. *Pattern-driven Model-based User-Interface Development*. 2007.
25. Jean sébastien Sottet, Gaelle Calvary, and Jean marie Favre. Towards mapping and model transformation for consistency of plastic user interfaces. In *The Many Faces of Consistency in Cross-Platform Design Workshop at CHI'2006*, 2006.
26. M. Scaife, Y. Rogers, F. Aldrich, and M. Davies. Designing for or designing with? Informant design for interactive learning environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 350. ACM, 1997.
27. Jean Vanderdonckt. A mda-compliant environment for developing user interfaces of information systems. In *Proc. of 17 th Conf. on Advanced Information Systems Engineering CAiSE'05*, pages 13–17. Springer-Verlag, 2005.