Udo Kannengiesser

Towards a Methodology for Flexible Process Specification

This paper proposes the foundations of a methodology for specifying process flexibility based on a view of processes as design objects. It is represented using the function-behaviour-structure (FBS) ontology of designing. The paper shows how the FBS ontology allows extending and generalising recent work on flexibility in engineering design, and how it allows applying this work to processes. The resulting framework provides a comprehensive account of process flexibility that subsumes existing approaches. Finally, the paper presents a method for flexible process specification, illustrated using examples of a property valuation process in the Australian lending industry.

1 Introduction

Flexible modelling of processes is a key issue for the effective use of process-aware information systems (PAIS) in dynamic business environments (Weber et al. 2009). Factors such as market or strategy changes, technological innovations and new regulations often require modifications of a process. Furthermore, unforeseen events in the immediate environment of the process need to be handled flexibly, such as resource bottlenecks or effects of unexpected human or system errors. PAIS using process models that are too rigidly specified are poorly applicable in real-world contexts and are ultimately rejected by their users.

Research has been concerned with understanding, modelling and implementing the notion of process flexibility. The taxonomies and methods resulting from these efforts address a wide range of aspects of flexibility (Daoudi and Nurcan 2007; Pesic and Aalst 2006; Regev et al. 2006, 2007). However, there is no coherent, comprehensive methodology as most approaches have been developed independently of each other. An attempt to providing such a framework has recently been undertaken by Schonenberg et al. (2008), proposing a general taxonomy of process flexibility associated with different realisation approaches.

Flexibility in PAIS is often viewed as a balance between the freedom to change and the need for stability (Regev et al. 2007). This balance is also an inherent characteristic of designing: Designers aim to change parts of the world through their designs, balancing the use of their individual perception and creativity, and the need to comply with requirements and constraints. A view of process performers as process re-designers has been well described by Aken (2007). This paper explores a design view of flexibility, aiming to establish broader, interdisciplinary foundations for understanding and specifying process flexibility. This provides the basis for a new methodology that augments existing frameworks of process flexibility.

Section 2 introduces an ontology of designing, the function-behaviour-structure (FBS) ontology, that can be applied to any object of designing, no matter whether this object is a physical product, a software product, or a process. The FBS ontology is then used to extend and generalise recent work on flexibility in engineering design. This provides the basis for a mapping between the design view of flexibility and existing approaches to modelling process flexibility. It is shown that none of the existing approaches encompasses and clearly distinguishes all aspects of flexibility. Section 3 develops the conceptual foundations for specifying process flexibility based on an existing framework of design activities, the situated FBS framework. A method is then derived and illustrated using a property valuation process in the Australian lending industry. Section 4 summarises and concludes the paper.

2 A Design View of Process Flexibility

2.1 The Function-Behaviour-Structure View of Designing

The FBS ontology (Gero and Kannengiesser 2004, 2007) is a well established way of conceptualising design objects. It has been applied to various instances of design objects, including physical products (Gero and Kannengiesser 2004), software (Kruchten 2005) and processes (Gero and Kannengiesser 2007).

Structure (S) is defined as a design object's components and their relationships. It can be viewed as the final outcome of a design process. In the domain of physical products, structure comprises the geometry, topology and material of individual components or assemblies. The structure of software consists of abstract constructs such as classes, components and pieces of code. In the domain of processes, structure includes three general classes of components: input, transformation and output (Gero and Kannengiesser 2007). The transformation often consists of a set of subtransformations, some of which can be viewed as 'micro-level' mechanisms that represent the 'materials' of the transformation. For example, a sequence of activities concerned with logging into an online banking system, and filling out and submitting a funds transfer form can be viewed as a process-centred 'material' of a payment transformation (Kannengiesser 2008). Other 'materials' are object-centred; they refer to the agent performing the transformation. Process-centred and object-centred 'materials' map onto Dietz' notions of realisation and implementation of a process, respectively (Dietz 2006). Structure encompasses control-flow, data, resource, and task views of a process (Kannengiesser 2008).

Behaviour (B) is defined as the attributes that can be derived from a design object's structure. They provide criteria for comparing and evaluating different design objects. An example of a physical product's behaviour is 'weight', which can be derived (or measured) from the product's structure properties of material and spatial dimensions. Behaviour of software (e.g., a text editor) includes its response time for visualising user input. It can be derived from software structure and its interaction with the operating environment. Typical behaviours of processes include speed, cost, precision and accuracy. They can be derived from process structure; for example, speed can be derived from (time-stamped) input and output.

Function (F) is defined as a design object's teleology ('what it is for'). This notion is independent of the common distinction between 'functional' and 'non-functional' properties; it comprises both as they describe the design object's usefulness for a stakeholder (or 'using system' (Dietz 2006)). This also distinguishes function from the concept of 'transfer function' (that corresponds to the transformation component of structure). Function is ascribed to behaviour by establishing a teleological connection between a human's goals and measurable effects of the design object. There is no direct connection between function and structure. The particular functions of a design object are ontologically independent of whether the design object's structure is conceptualised as a physical product, a software product or a process. For example, the functions 'wake people up', 'be reliable' and 'be punctual' may be ascribed to relevant behaviours of a mechanical alarm clock (i.e., a physical product), a virtual alarm clock (i.e., software), or a sequence of activities (i.e., a process).

From a high-level perspective, designing can be viewed as decision making. This view implies the existence of choices (Gero 1994) that can be represented as alternative values for the variables of the design. The set of all design variables and their ranges of values form what is called the design state space, i.e., the space of all possible designs. The design state space is partitioned into three subspaces: function state space, behaviour state space, and structure state space, as shown in Fig. 1. The three subspaces are interconnected through the designer's compiled knowledge of qualitative and quantitative relationships between function, behaviour and structure. These relationships are the basis for modelling designing as an activity that aims to produce structure that exhibits suitable behaviour to which desired function can be ascribed.

The notion of a design state space allows understanding designing through the use of spatial metaphors. Selecting values for a set of design variables can be described as a process of moving through the design state space. This model of designing is often referred to as *search* (Gero 1994). However, most designing involves more than moving through a well-defined space of known design alternatives. It also involves generating the space in terms of design variables and their ranges of values. This can involve discarding some previously expected variables or ranges of values, leading to a shift of the design state space. The notion that addresses these changes is called *exploration* (Gero 1994). Changes may affect all three subspaces, and changes of one subspace may lead to changes of a subspace connected to it. For example, a change of the structure state space may lead to changes of the behaviour state space. This, in turn, may lead to subsequent changes of the function state space and/or the structure state space.

Expectations about the design problem are fundamental in distinguishing exploration from search. Exploration reflects changed expectations as the designer learns more about the design problem by interacting with it (Schön 1983). In contrast, the notion of search reflects unchanged expectations of the design (state space). Some of the initial design expectations are formulated through explicitly stated requirements. Others arise from the designer's understanding of the design object's socio-technical environment across the life cycle. The possible change of a design state space from its inception to a later point in time is presented conceptually in Fig. 2. The increasing size of the design state space is to indicate that a great deal of the knowledge required to produce a design is constructed during designing (Logan and Smithers 1993).

The application of the state space concept in PAIS has commonly had a more narrow focus. It is usually understood only as a representation of the set of possible changes in the world that may occur during the execution of a process instance, not as a representation of the set of all possible process designs. Recent work on business rules in process models (Dietz 2008) can be viewed as expanding the scope of state space models of processes. However, these approaches are limited to the notion of a structure state space, and do not cover behaviour and function state spaces.

Design state spaces can be represented in many ways, most of which can be classified as one of the following approaches:

- Enumeration (e.g., a set of alternative product modules (Greer et al. 2003) or process fragments (Sadiq et al. 2001)),
- Generative rules (e.g., grammars (Brown 1997)),
- Constraints (e.g., in optimisation models (Papalambros and Wilde 2000), or declarative process definitions (Pesic and Aalst 2006)), and
- Abstraction (e.g., using types defined in domain ontologies or taxonomies (Gorti et al. 1998)).

Table 1 shows approaches that are most commonly used for representing function, behaviour and structure subspaces.

2.2 Generalising an Engineering Design Approach to Flexibility

Flexibility has been a popular concept in many areas within engineering design. Similar to process flexibility, it is understood here as the ability

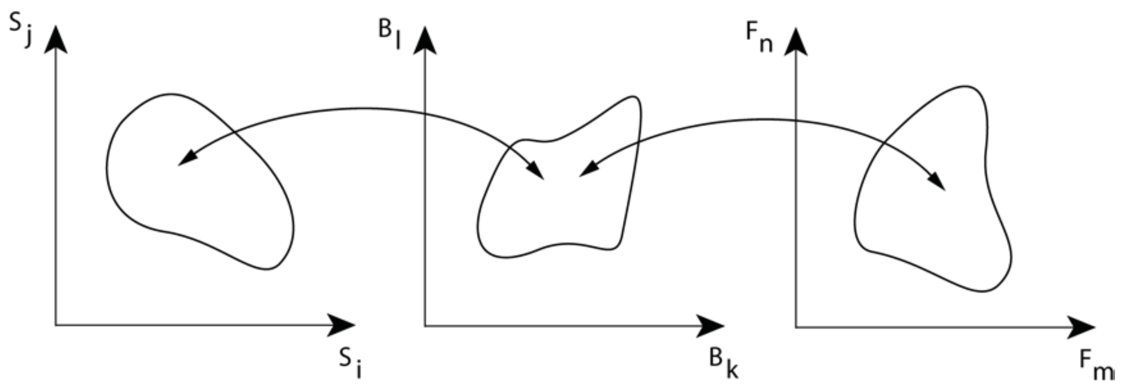

Figure 1: Function, behaviour and structure state spaces, and their interconnections

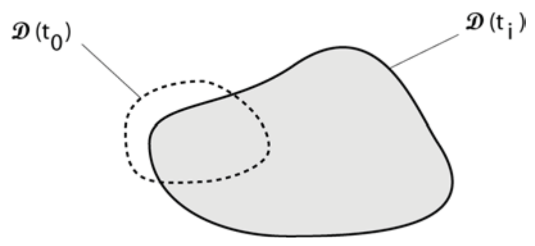

Figure 2: A design state space (D) changes between the initial time t_0 and a later time t_i

of a product or system to handle change. However, more precise definitions of this notion are often missing (Saleh et al. 2003). One of the most comprehensive approaches to defining and characterising flexibility in engineering design has recently been proposed by researchers from the MIT Engineering Systems Division (Ross et al. 2008). This Section provides an overview of relevant concepts of this work, and expands and generalises them using the FBS ontology. Ross et al. (2008) propose two aspects of flexibility of a design object: change effects, and change mechanisms.

Change effects characterise the difference between the states of a design object before and after its change. States are described in terms of variables and values that may refer to any aspect of the design object, including function, behaviour and structure. There are three categories of change effects: robustness, scalability, and modifiability.

Robustness is the ability to maintain the design object's required functions without changing its structure, despite the presence of changes affecting the object's internal or external environment (Ross et al. 2008). For example, a car may achieve its function of transportation without changing its design, despite internal changes such as tire abrasion or external changes such as altered road conditions. Robustness handles change by being insensitive to it. In fact, it has been understood as a concept that is related but quite distinct from flexibility (Saleh et al. 2003).

Scalability is the ability to vary the design object's state in terms of the values of its variables (Ross et al. 2008). For example, varying the length, width and height of a mobile phone is a scalable change of structure. Varying the speed of a central processing unit is a scalable change of behaviour. And varying the reliability of an alarm clock is a scalable change of function. Scalability is captured in the state space representation of designing as either search (if the change remains within state space boundaries) or exploration (if the change involves crossing a state space boundary in terms of ranges of values).

Table 1: Common approaches for representing FBS subspaces

Subspace	Common representation approaches
F state space	Enumeration, abstraction
B state space	Constraints
S state space	Enumeration, generative rules, constraints, abstraction

Modifiability is the ability to vary the design object's state in terms of its variables (Ross et al. 2008). For example, the addition of a DVD burning module to a computer is a modifiable change of structure. Changing a car's petrol consumption rate to rapeseed oil consumption rate is a modifiable change of behaviour. And augmenting a mobile phone with the ability to play MP3 files is a modifiable change of function. Modifiability is captured in the state space representation of designing as exploration via changing the set of variables.

Variations of function, behaviour and structure rarely occur in isolation of one another. As outlined in Sect. 2.1, a change of one subspace often leads to changes of other subspaces. The specific ways in which a change propagates across the subspaces depend not only on given requirements and constraints but also on the individual expertise and interpretations of the designer. We can view this as an additional dimension of flexibility, embedded within the notion of change effects.

Change mechanisms represent different ways of achieving the desired change effects. Ross et al. (2008) propose the number of possible change mechanisms, filtered by a subjective acceptability threshold for their 'cost', as a basis for quantifying flexibility. Here, 'cost' is an aggregated measure for the consumption of various resources including time and money.

We can expand the notion of change mechanisms by defining three categories, Fig. 3: design goal achievement, design realisation, and design assessment.

Design goal achievement is the ability to vary the activities and resources required for transforming intended changes of function (ΔF_i) into intended changes of structure (ΔS_i) via intended changes of behaviour (ΔB_i). For example, besides generating design changes from scratch, one may have the option of reusing previous design knowledge captured in patterns, best practices, rationale, case bases, prototypes or other forms of representation. Each of these options requires different technologies and user skills. Design goal achievement also includes strategies with which one can search for or explore appropriate change effects in the function, behaviour and structure state spaces.

Design realisation is the ability to vary the activities and resources required for transforming an intended change of structure (ΔS_i) into a realised change of structure (ΔS_r). This includes the allocation of agents (machines or people), their coordination and any setup tasks required (for example, programming, instructing and training). It also includes strategies for preventing or mitigating issues, such as downtime and obsolete work items arising from changes in the realisation.

Design assessment is the ability to vary the activities and resources required for monitoring, analysing and validating the success (for example, the consistency, correctness and efficiency) of the change. This includes methods and tools available for deriving changes of behaviour (ΔB_r) from a realised change of structure (ΔS_r), and ascribing changes of function (ΔF_r) to these changes of behaviour.

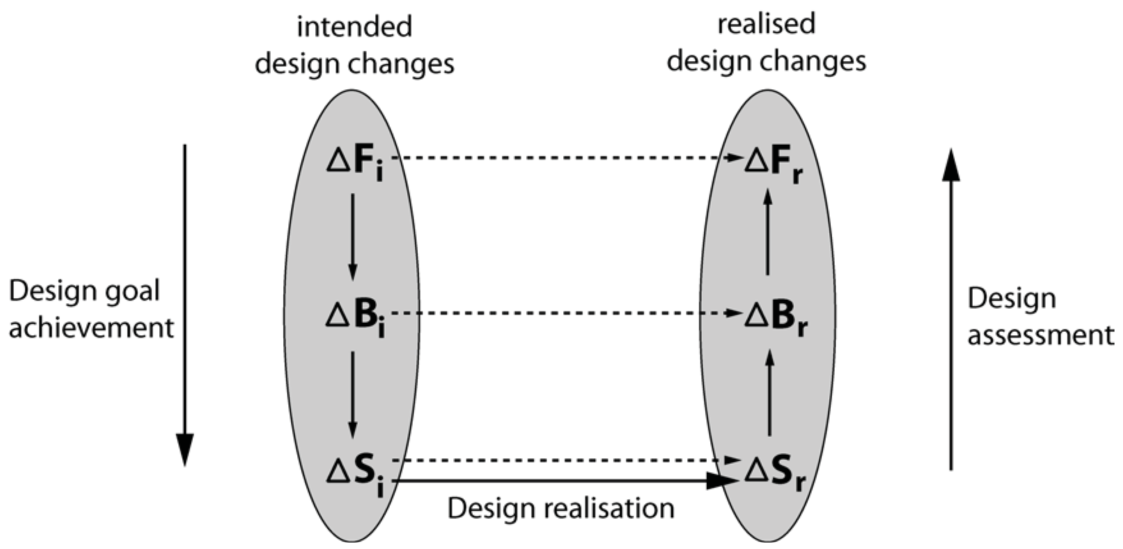

Figure 3: Three categories of change mechanisms: design goal achievement, design realisation, and design assessment.

2.3 Casting Existing Approaches to Process Flexibility in a Design View

The three categories of change effects (robustness, scalability, and modifiability) and the three categories of change mechanisms (design goal achievement, design realisation, and design assessment) can be mapped onto existing research in flexible PAIS.

Robustness captures 'flexibility by design' (Schonenberg et al. 2008) or 'flexibility by definition' (Sadiq et al. 2001) that is the ability to include multiple execution paths in the process model at design time. They represent different ways of dealing with anticipated variations in the internal or external environment of the process. The different paths are selected at runtime for individual process instances.

Scalability captures two categories of process flexibility proposed by Schonenberg et al. (2008) that imply the existence of expected choices. One is 'flexibility by deviation' (Schonenberg et al. 2008) that is the ability of a process instance to deviate from the original process model (type) without altering it. It encompasses only changes in the execution sequence of tasks, not the tasks themselves. We conceptualise the ordering relationships that determine the execution sequence as a set of interrelated 'ports' of the tasks (Gottschalk et al. 2008). Specifically, every task has variables for their 'inflow ports' and 'outflow ports', and the values of these variables are pointers to other tasks. Changing the relationships can then be viewed as varying the values of structure variables. The other category of process flexibility corresponding to scalability is 'flexibility by underspecification' (Schonenberg et al. 2008) or 'flexibility by templates' (Sadiq et al. 2001) via late binding of process fragments to a placeholder. This category of process flexibility is the ability to execute an incomplete process model by completing it at runtime, via selection from a pre-defined set of process fragments. The fragments can be represented as structure variables with Boolean values. A process fragment with the value 'false' means that this fragment is currently not selected. Changing the value to 'true' corresponds to selecting it to instantiate the placeholder. Potential subjects of scalable change include not only control flow but also other aspects of process structure (Regev et al. 2006).

Scalable changes of process function and process behaviour are not included in existing work on process flexibility. Examples include improving maintainability of a process (a scalable change of function), and reducing the cost of a process (a scalable change of behaviour).

Modifiability captures two categories of process flexibility proposed by Schonenberg et al. (2008) that imply a shift of expectations. One is 'flexibility by underspecification' (Schonenberg et al. 2008) via late modelling, which is the ability to construct a new process fragment for a placeholder. It is best thought of as the generation of a new variable to be introduced in the structure state space of the process. The other category mapping onto modifiability is 'flexibility by change' (Schonenberg et al. 2008), which is the ability to modify a process at runtime, in response to unforeseen circumstances in the execution environment. Here, changes represent new tasks being introduced and/or removed, affecting process instances and/or process types. By representing every task as a structure variable, we can model these changes as modifications of the structure state space. Potential subjects of modifiable change include not only control flow but also other aspects of process structure, such as informational, organisational and operational aspects (Regev et al. 2006).

Modifiable changes of process function and process behaviour are not included in most existing work on process flexibility. Examples include considering the waste production of a manufacturing process in addition to other process attributes (a modifiable change of behaviour), and changing the goal of a transportation process from 'people transportation' to 'cargo transportation' (a modifiable change of function). There is work on using variations of quality goals to generate different configurations of process structure (e.g., Lapouchnian et al. (2007)).

Design goal achievement is addressed by the range of methodologies for process design. Methodologies differ in their notations, their coverage of different process views, their technological support, and their ease of use. A number of existing technologies, including ADEPT1, YAWL, FLOWer and Declare, have been examined by Schonenberg et al. (2008) for their potential use as mechanisms to achieve different change effects.

Design realisation subsumes migration strategies for running process instances, and mechanisms for version, access and concurrency control, which are described by Weber et al. (2008). Design realisation also includes methods and technologies for communication and 'setup' (instructing, training, etc.), and strategies for adapting to variations in the execution environment (including some exception-handling strategies).

Design assessment includes methods and technologies for analysing correctness, consistency, efficiency, traceability, usability and other process quality attributes (Rinderle et al. 2004; Weber et al. 2008).

Casting existing frameworks for classifying process flexibility in a design view shows that they fall short in two ways: First, there is no single approach that captures both change effects and change mechanisms. The framework proposed by Schonenberg et al. (2008) covers all categories of change effects, including robustness, scalability and modifiability, but largely excludes change mechanisms. In turn, the work by Weber et al. (2008) strongly focuses on change mechanisms, but does not address change effects. Other approaches incorporate parts of both aspects but without clearly differentiating between them. For example, the taxonomy proposed by Regev et al. (2006) contains 'abstraction level' and 'subject' of change, which relate to change effects, and 'properties' of change (e.g., 'extent', 'duration', and 'swiftness'), which relate to attributes of change mechanisms. Similarly, the classification of process changes proposed by Aalst and Jablonski (2000) is a collection of features describing change effects (e.g., 'perspectives', and 'kinds of change') and change mechanisms (e.g., timing of change, and migration strategies), without explicit distinction of the two aspects.

The second shortcoming of most existing approaches is that they do not explicitly address the notions of function and behaviour. There are some existing characterisations that may be cast in these notions. For example, some the 'reasons for change' proposed by Aalst and Jablonski (2000) can be interpreted in terms of the function or behaviour that underpin a particular change of process structure. The categories of 'type flexibility', 'volume flexibility' and 'structural flexibility' proposed by Snowdon et al. (2007) are derived from influencing factors for flexibility that may also be mapped onto function and behaviour. In the broader area of process modelling, there is an increasing interest in integrating notions of goals (Lapouchnian et al. 2007; Soffer and Regev 2005) and 'context' (Rosemann et al. 2008) into process models to more comprehensively capture how changes of process structure can be traced back to their underlying rationale. The Worklets approach (Adams et al. 2006) aims to realise flexible process execution through context-aware selection of process fragments at runtime. However, existing frameworks of process flexibility do not provide a generic schema for explicitly representing process function and behaviour besides process structure, thus restricting the scope of flexibility in PAIS.

3 Specifying Process Flexibility

3.1 Conceptual Foundations

The design view of process flexibility presented in Sect. 2 is based on the idea of a design state space with its three subspaces for function, behaviour and structure. We have presented this notion as the set of all possible designs based on the expectations and experience of the individual designer. When we adopt a prescriptive stance, a design state space becomes the set of all 'permitted' designs according to specifications given to the process designer. Here, the boundaries of the design state space, both in terms of the specified set of variables and their ranges of values, represent requirements that may be socially enforceable. In fact, the specified design state space represents a 'normative restriction of design freedom' (Dietz 2006).

On the other hand, as shown in Fig. 2, parts of the initial specification of a design state space may be relaxed over the course of designing, resulting in a new design state space with modified boundaries. Therefore, different degrees of 'normative strength' of the state space boundaries should be made explicit to specify what parts of the space may be changed and what parts must not. This can be realised by associating individual design variables and their ranges of values with modality attributes such as 'mandatory' and 'optional'. Figure 4 shows that specifying a subset D_m of an initial design state space $D(t_0)$ as mandatory requires any subsequent design state space $D(t_i)$ to also include these parts. The relative complement of D_m with respect to $D(t_0)$ is the set of optional requirements that may or may not be included in $D(t_i)$.

The approach to understanding flexibility developed in Sect. 2 can be cast in an existing framework of designing, the situated FBS framework (Gero and Kannengiesser 2004). Its basis is a three-world model of interactions in designing, Fig. 5(a). The *external world* is composed of representations outside the designer or design agent. The *interpreted world* is built up inside the design agent in terms of sensory experiences, percepts and concepts. It is the internal representation of that part of the external world that the design agent interacts with. The *expected world* is the world imagined actions of the design agent will produce. It is the environment in which the effects of actions are predicted according to current goals and interpretations of the current state of the world.

These three worlds are linked together by three classes of connections. *Interpretation* transforms variables which are sensed in the external world into the interpretations of sensory experiences, percepts and concepts that compose the interpreted world. *Focussing* takes some aspects of the interpreted world, and uses them as goals for the expected world that then become the basis for the suggestion of actions. *Action* is an

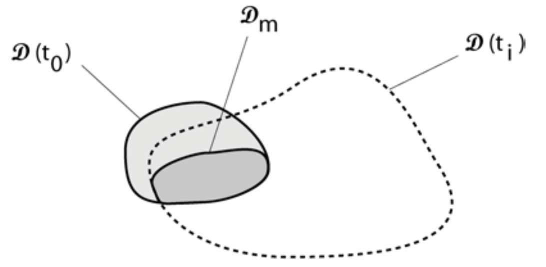

Figure 4: A set of mandatory requirements D_m can be specified that is a subset of the initial design state space $D(t_0)$ and all subsequent design state spaces $D(t_i)$

effect which brings about a change in the external world according to the goals in the expected world.

The two kinds of design changes presented in Fig. 3, i.e., intended and realised design changes, can be mapped onto this three-world model. Intended design changes are goals that direct the execution of actions, and are therefore included in the expected world. Realised design changes are concepts that have been interpreted for assessment, and are therefore included in the interpreted world. Intended and realised design changes are linked via the external world that includes the realisation of intended changes.

Figure 5(b) specialises the three worlds by nesting them and articulating general classes of design representations as well as the activity of reflection (Schön 1983). The set of expected design representations (Xe^i) corresponds to the notion of a design state space. This state space can be modified during designing by transferring new interpreted design representations (X^i) into the expected world and/or transferring some of the expected design representations (Xe^i) out of the expected world. This leads to changes in external design representations (X^e), which may then be used as a basis for re-interpretation changing the interpreted world. Novel interpreted design representations (X^i) may also be the result of *constructive memory*, which can be viewed as an interaction among design representations within the interpreted world rather than across the interpreted and the external world. Both interpretation and constructive memory are represented as 'push-pull' activities (Gero and Fujii 2000). This emphasises the role of individual experience in constructing the interpreted world, by 'pulling' interpreted representations rather than just by 'pushing' what is presented in the external world. It is the interaction of push and pull that may produce new representations that can be used to modify the design state space.

A specific set of external design representations are requirements (XR^e). They can be thought of as specifying an initial design state space at time t_0, as represented in Fig. 2. XR^e consists of mandatory requirements XR^e_m and optional requirements XR^e_o, i.e., $XR^e = XR^e_m \cup XR^e_o$ with $XR^e_m \cap XR^e_o = \emptyset$. We assume for the purposes of this paper that all requirements are unambiguous and fixed, and that they match their representation in the interpreted world at an initial time t_0, i.e., that $X^i(t_0) = XR^e$. Further, we assume that the complete set of interpreted requirements is transferred into the expected world at the same time t_0, i.e., that $Xe^i(t_0) = X^i(t_0) = XR^e$. These assumptions simplify the roles of interpretation and focussing activities to simple one-to-one mappings.

When there are changes of the design state space at a later point in time t_1, the new design state space $Xe^i(t_1)$ includes additional representations (through additional variables or expanded ranges of values) or drops initially focused representations (through eliminated variables or reduced ranges of values), i.e., $Xe^i(t_1) \Delta Xe^i(t_0) \neq \emptyset$. However, the modified design state space must include all mandatory requirements XR^e_m,i.e.,

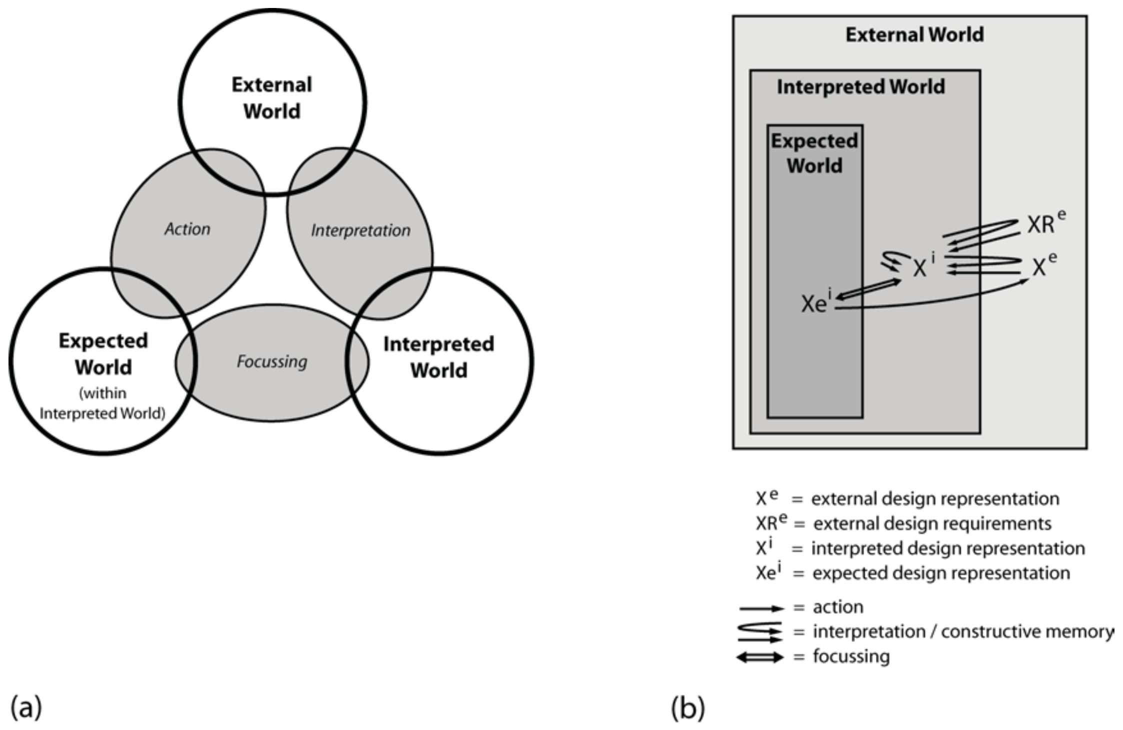
External World
Action
Interpretation
Expected World (within Interpreted World)
Focussing
Interpreted World
(a)
External World
Interpreted World
Expected World
XR^e
X^i
X^e
Xe^i
X^e = external design representation
XR^e = external design requirements
X^i = interpreted design representation
Xe^i = expected design representation
= action
= interpretation / constructive memory
= focussing
(b)

Figure 5: Three interacting worlds: (a) general model, (b) specialised model for design representations

$Xe^i(t_1) \supseteq XR^e_m$. Specifying a modifiable design state space enables the potential for generating changes of the design (state space) via focussing.

The activities enabled by the permitted change effects can be represented in more detail using the situated FBS framework (Gero and Kannengiesser 2004), presented in Fig. 6. This framework integrates the model in Fig. 5(b) with the concepts described in Fig. 3, by replacing the general design representation X in Fig. 5(b) with the more specific design representations of F, B and S, and by adding activities that transform F into B, and B into S in the expected world, and vice versa in the interpreted world. The situated FBS framework defines 20 labelled activities in designing (note that the labels do not represent any order of execution; see Gero and Kannengiesser (2004)). These 20 design activities represent general classes of change mechanisms. External requirements are grouped into requirements on function (FR^e), behaviour (BR^e) and structure (SR^e). As a result, the initial design state space consists of $Fe^i(t_0) = F^i(t_0) = FR^e$, $Be^i(t_0) = B^i(t_0) = BR^e$, and $Se^i(t_0) = S^i(t_0) = SR^e$, formed by interpretation (activities 1 to 3 in Fig. 6) and focussing (activities 7 to 9).

Table 2 provides a summary of the activities (or change mechanisms) enabled by different specifications of function state spaces, behaviour state spaces and structure state spaces. The activities are categorised as either exploration or search activities (as introduced in Sect. 2.1). Exploration captures activities used for reformulating the initial design state space. They include two types of activities: those that directly change the expected world (activities 7 to 10), and those that generate new representations as precursors of changing the expected world (activities 4, 5, 6, 13, 14, 16, 19, 20; for more information see Gero and Kannengiesser (2004, 2007)). Search captures activities used for selecting structure instances within the structure state space (activity 11), realising them (activity 12), and testing them based on expected behaviours (activities 13 to 15). Itera-

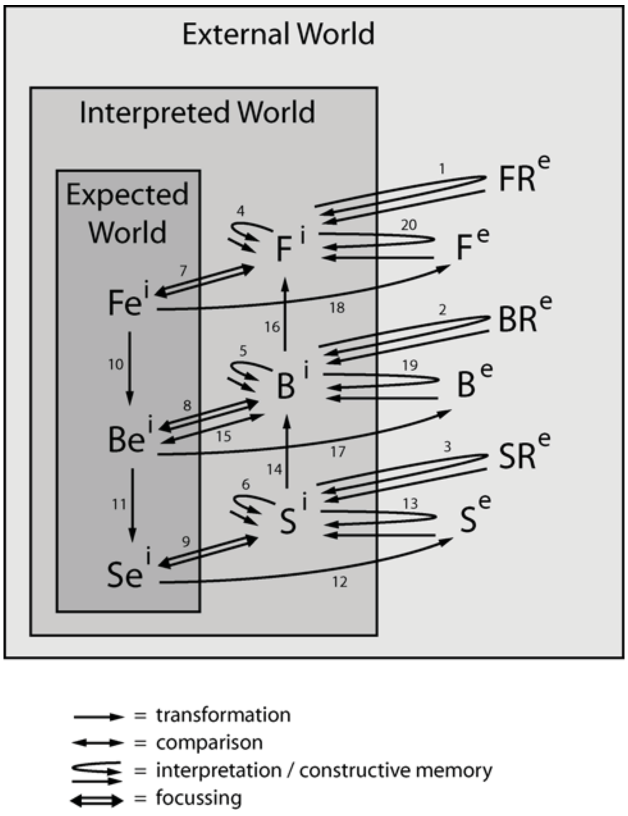

Figure 6: The situated FBS framework

tions may occur as these activities can be carried out multiple times.

We can subsume the different specification types presented in Tab. 2 in the following general scenarios:

a) *Specification of S robustness*: This is the case where the initial structure state space contains only one point that is specified as mandatory. As a result, the process structure can be realised (activity 12) immediately without considering any alternative structures. The external structure of the process, often in the form of one or more process executions, can be monitored or interpreted (activity 13), and its behaviour can be derived (activity 14) and then compared against expected behaviour (activity 15). In case of unsatisfactory results of this assessment, only the external process structure can be changed (via activity 12), not the expected process structure. The difference between the external and expected structure is that the latter is often an ex post rationalisation (Weick 1995) of the former, by pruning some external components (or process steps) and their relationships (or control flows). High-level, linear structures can thus be interpreted from detailed, iterative external process structures. The difference between expected and external structures can be understood as the difference between the notions of process architecture and process realisation (Kannengiesser 2009), respectively.

b) *Specification of S scalability*: Specifying a scalable process structure enables activity (11) that selects among different process structures to find one that meets expectations of behaviour. All candidate structures must be within the specified structure state space. Once a particular structure is realised (activity 12), it is in-

Table 2: Specification types based on different change effects enable different design activities. Numbers refer to the labelled activities in the situated FBS framework.

External requirement	Specification type	Exploration	Search
SR^e	S-robustness	---	12, 13, 14, 15
	S-scalability	---	11, 12, 13, 14, 15
	S-modifiability	9, 6, 13	
BR^e	B-scalability	exploration activities enabled by S-modifiability	
	B-modifiability	8, 5, 19, 14, 10; and exploration activities enabled by S-modifiability	
FR^e	F-scalability	exploration activities enabled by B- and S-modifiability	
	F-modifiability	7, 4, 20, 16; and exploration activities enabled by B- and S-modifiability	

terpreted (activity 13), and its behaviour is derived (activity 14) and then compared against expected behaviour (activity 15). If the structure does not meet expectations, it can be replaced with another structure within the same state space (activity 11). This scenario subsumes the one described in (a) by opening up the potential of changing, within specified ranges, the expected structure rather than only the external structure.

c) *Specification of S modifiability*: Specifying a modifiable process structure enables changes of the initial structure state space through addition, subtraction or substitution of variables, or through modifying ranges of values (activity 9). This allows generating and testing new classes of process structures (activities 11 to 15) that were not possible using the initial structure state space. There are two drivers for these changes: interpretation of previously generated (executed) process structures (activity 13), and constructive memory (activity 6). The former can produce new variables because it has the potential for re-conceptualising process structures. Examples include the discovery of workflows, which has been described in literature on process mining (Aalst and Weijters 2004). This scenario subsumes the one described in (b) by enabling exploration of expected structures in addition to search within these structures.

d) *Specification of B- or F- scalability or modifiability*: Specifying flexibility at the levels of function or behaviour constrains process structures only indirectly, through the interconnections of the three subspaces based on individual experience. For example, a scalable or modifiable change of behaviour may be achieved with the same structure (S robustness), or with alternative structures within (S scalability) or outside (S modifiability) the initial structure state space. The same holds for scalable and modifiable changes of function; they may be achieved with alternative behaviours within (B scalability) or outside (B modifiability) the initial behaviour state space. Specifying B modifiability enables focussing on different behaviours (activity 8), and specifying F modifiability enables focussing on different functions (activity 7). Providing details about the drivers for these reformulations (activities 4, 5, 10, 14, 16, 19 and 20) is beyond the scope of this paper; interested readers may refer to Gero and Kannengiesser (2007). This scenario subsumes the one described in (c).

Making a process specification more flexible by moving its type towards the bottom of Tab. 2 increases the agent's 'design freedom' as it enables a higher number of design activities. While this can be beneficial in many circumstances, it may also have undesired consequences arising from the *ad hoc* way in which changes may now be carried out. These consequences can be controlled by constraining the enabled design activities (i.e., the general classes of change mechanisms), again by using a set of FBS requirements. We refer to these requirements as *mechanism requirements*, distinguishing them from the requirements on change effects, which we call *effect requirements*. We will use this distinction in the next Section that presents as method for specifying flexible processes using both kinds of requirements.

3.2 A Method for Specifying Flexible Processes

The conceptual framework developed in Sect. 3.1 provides a basis for a new method for specifying flexible processes. This method includes constructs that can cover the complete spectrum of possible specification types shown in Tab. 2. Traditional, flow-based process modelling languages can cover only the specification of structure robustness. Their usefulness is generally limited to coarse-grained process definitions. Attempts to specify robustness in more detailed process structures often fail due to increasing heterogeneity and dynamics of the process environment. Approaches based on the notions of scalability and modifiability, such as the ones evaluated by Schonenberg et al. (2008), have shown their benefits in dynamic environments. They can be included in a more comprehensive methodology that applies scalability and modifiability not only to specifying structure but also behaviour and function.

The method consists of three basic steps:

1. **Specify robust process structure**: Specify a process structure in a top-down manner using a flow-based modelling language, up to a level of detail where possible process variations are sufficiently known and can be captured in the process model. The result is a process structure that is robust at this level of detail.
2. **Specify scalable and modifiable change effects**: Identify the individual process activities that need to be elaborated, define requirements on their function (FR^e), behaviour (BR^e) and/or structure (SR^e), and connect these effect requirements as annotations to the process activities. Effect requirements are specified for every process activity by associating every element of a subspace representation with a modality. Approaches for representing FBS subspaces may be chosen based on Tab. 1. Modalities are either 'mandatory' or 'optional', or may include more fine-grained attributes such as proposed by Borch and Stefansen (2006).
3. **Specify change mechanisms**: Identify, using Tab. 2, those enabled design activities (or change mechanisms) that need to be constrained, define requirements on their function, behaviour and/or structure, and reference these mechanism requirements from the effect requirements. References linking the two requirements are established based on the output of every activity (indicated by the arrowheads in Fig. 6): Mechanism requirements on activities whose outputs are function, behaviour or structure, are referenced by the effect requirements FR^e, BR^e or SR^e, respectively. In particular, FR^e may include references to the mechanism requirements on activities 7, 4, 20 and 16; BR^e can include references to the mechanism requirements on activities 8, 5, 19, 14, 15 and 10; and SR^e can include references to the mechanism requirements on activities 9, 6, 11, 12 and 13.

We can illustrate the basics of this method by applying it to a property valuation (short: valuation) process in the Australian lending industry.

Step 1 (Specify robust process structure) produces a top-level BPMN model of process structure, Fig. 7, that is assumed to be robust at this level of detail. The process starts when the valuation company receives a request from a lender (e.g., a bank) to assess the market value of a specific property. An employee (called the 'valuer') is then assigned to perform the valuation by inspecting the property and preparing a valuation report that contains the estimated market value of the property. After that, the valuation report is sent to the lender, and, concurrently, an invoice is sent. Upon receipt of payment, the valuation process terminates.

Figure 7 includes three collapsed sub-processes, one of which ('Perform Valuation') is expanded as shown in Fig 8. This is again assumed to be a robust process structure, at a more detailed level. The other two sub-processes ('Assign Valuer' and 'Send Valuation Report') are not expanded due to uncertainty regarding the process environment and consequently the specific process variations likely to be needed.

Step 2 (Specify scalable and modifiable change effects) defines a set of FBS requirements for all three sub-processes, and includes them in the model via annotations, Fig. 9. The approaches for representing FBS subspaces are chosen from Tab. 1 and include enumeration for representing function state spaces, constraints for representing behaviour state spaces, and abstraction for representing structure state spaces. The rationale underpinning some of these choices is easy comprehension for readers of this paper. In particular, this has led to choosing abstraction for representing structure state spaces, despite the lack of a well-defined, formal ontology for the domain of property valuation. It is clear that a more formal approach, such as declarative (constraint-based) notations, would be required when defining specifications for process execution and analysis.

Step 3 (Specify change mechanisms) identifies those enabled design activities (change mechanisms) that need to be constrained in addition to the change effects specified so far. Figure 10 shows the results of this Step: A mandatory 'migration type = proceed' (see Schonenberg et al. (2008)) is added to the requirements on 'Assign Valuer', and a mandatory 'exception-handling type = deferred fixing' (see Lerner et al. (2010)) is added to the requirements on 'Perform Valuation'. They establish new mechanism requirements that are added to the existing effect requirements. As 'migration type' and 'exception-handling type' both relate to the realisation of structure (design activity 12 in Fig. 6), they are attributed to the structure (S) parts of the annotations.

How do the process specifications obtained through applying our method affect process flexibility? Let us have a look at the sub-process 'Perform Valuation'. Its expected (and required) structure is specified as robust, as shown in Fig. 8, with a requirement on the realisation of this structure, related to exception handling. This may result in an external structure, as described in Fig. 11, that includes additional activities and paths for detecting and dealing with exceptional situations in the realisation of the sub-process. The activity 'Visually Inspect' is added to detect complicated site conditions, such as slopes and irregular building shapes. These exceptional circumstances are immediately taken note of ('Record Nature of Complications'), but their consequences are dealt with ('Renegotiate Fees') later in the sub-process. This 'deferred fixing' exception-handling strategy is time-efficient and a likely way in which the required behaviour of the sub-process, 'time < 2 business days' (see Fig. 10), can be achieved. This strategy allows scalable changes of realising external structure (design activity 12 in Fig. 6) in that the 'Renegotiate Fees' activity may also be carried out either immediately prior to or in parallel with the 'Prepare Valuation Report' activity. The kind of flexibility presented in this example is thus consistent with scenario (a) in Sect. 3.1.

The structure state space specified by the annotation of 'Assign Valuer' in Fig. 10 includes a family of process fragments that can all be characterised as 'top-down selection' processes. One of these fragments needs to be selected (design activity 11), based on the expected (required) behaviour of 'time < 2 business days'. Figure 12 shows two 'top-down selection'- type process fragments of 'Assign Valuer'. Let us assume that

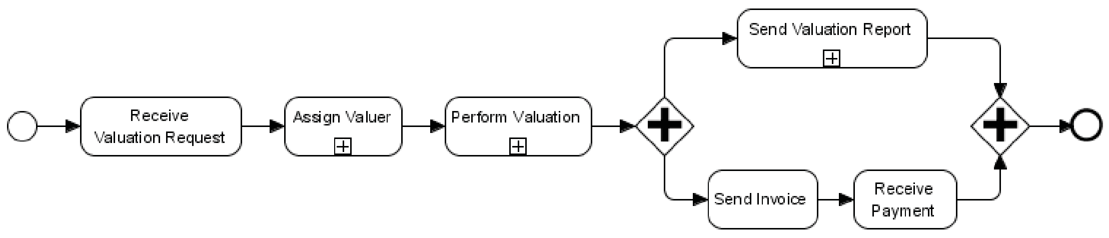

Figure 7: Step 1 produces a robust structure of a property valuation process at the top level

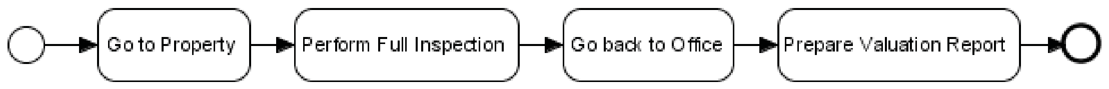

Figure 8: Step 1 produces a robust structure of the sub-process 'Perform Valuation'

Fragment 1 (Fig. 12(a)) is selected as a candidate structure within the structure state space (Se^i). After realising, interpreting and assessing it (design activities 12 to 15), the agent may decide to enhance time behaviour by selecting Fragment 2 (Fig. 12(b)) as the new candidate structure and again testing the resulting behaviour. The design activities enabled in this example are consistent with scenario (b) in Sect. 3.1.

Since the 'top-down selection' type of process fragments is specified as optional rather than mandatory, the agent may make more significant changes by modifying the structure state space. Figure 13 shows a process fragment that is selected (design activity 11) from a new 'bidding'-type family of process structures, replacing the previous 'top-down selection' type (design activity 9). With this 'bidding'- type structure, the sub-process can quickly identify those potential valuers that are currently located near the property to be valuated. As a result, the time for assigning valuation jobs can be significantly reduced. The new structure is inspired by the way taxi companies dynamically assign incoming customer requests to specific drivers. This can be modelled as the interpretation (design activity 13) of an external process structure from a different domain. The modifiable change of expected structure illustrated by Fig. 13 requires further changes that are not shown in the BPMN model, such as the development and implementation of appropriate information and communication technologies. Once the new expected structure is realised (design activity 12), it needs to be assessed in a similar way as described earlier (design activities 13 to 15). The design activities enabled in this example are consistent with scenario (c) in Sect. 3.1.

The annotation of 'Send Valuation Report' in Fig. 10 includes only specifications of functions, leaving a great deal of design freedom in terms of the design activities needed to achieve the required functions. The agent can use domain knowledge to transform the function 'provide interoperable data' into a new behaviour (design activity 10) that may be termed 'LIXI compliance'. LIXI (Lending Industry XML Initiative; see www.lixi.org.au) is a consortium of the Australian lending industry with the goal of introducing interoperable, straight-through processing of mortgage applications based on the standard Credit Application Language (CAL). LIXI compliance can be achieved by using new expected structure variables including 'output file type = XML' and 'output vocabulary type = CAL'. These new variables are constructed internally using domain knowledge (design activity 6) and then introduced in the structure state space through focussing (design activity 9). The design activities enabled in this example are consistent with scenario (d) in Sect. 3.1.

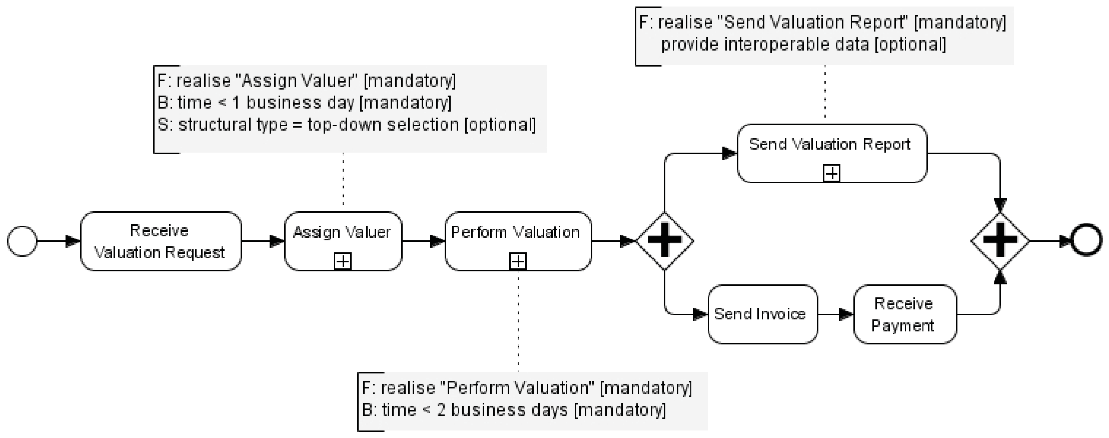
F: realise "Send Valuation Report" [mandatory]
provide interoperable data [optional]
F: realise "Assign Valuer" [mandatory]
B: time < 1 business day [mandatory]
S: structural type = top-down selection [optional]
Send Valuation Report
Receive Valuation Request
Assign Valuer
Perform Valuation
Send Invoice
Receive Payment
F: realise "Perform Valuation" [mandatory]
B: time < 2 business days [mandatory]

Figure 9: Step 2 produces annotations that contain effect requirements for individual sub-processes

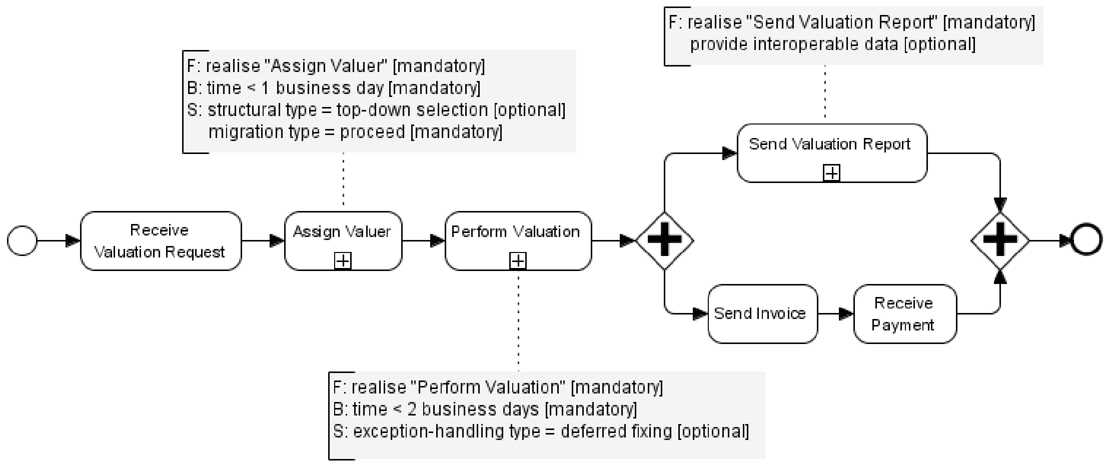
F: realise "Send Valuation Report" [mandatory]
provide interoperable data [optional]
F: realise "Assign Valuer" [mandatory]
B: time < 1 business day [mandatory]
S: structural type = top-down selection [optional]
migration type = proceed [mandatory]
Send Valuation Report
Receive Valuation Request
Assign Valuer
Perform Valuation
Send Invoice
Receive Payment
F: realise "Perform Valuation" [mandatory]
B: time < 2 business days [mandatory]
S: exception-handling type = deferred fixing [optional]

Figure 10: Step 3 adds mechanism requirements to the existing effect requirements

4 Conclusion

A methodology for flexible process specification that is based on a design view of process flexibility leverages a characteristic that is inherent in the nature of designing: the capacity to operate within a space of alternatives that is generated based on not only a set of requirements but also the designer's individual understanding of the problem. The methodology presented in this paper expands and generalises a recent approach from engineering design, using a domain-independent ontology of designing. Current approaches to modelling process flexibility fit in this framework, but do not explicitly cover the notions of function and behaviour. These aspects capture 'what should be done without specifying how it should be done' (Pesic and Aalst 2006) in a more meaningful and comprehensive way than existing declarative approaches that are limited to process structure. The design view provides a unifying framework that brings together different research streams in flexible PAIS, most of which can be categorised as focusing on either change effects or change mechanisms.

The proposed methodology can be used to specify the flexibility of a process at different design-ontological levels and with different degrees of 'normative strength'. It supports the view of stakeholders as designers or re-designers of the process, while constraining their design activities

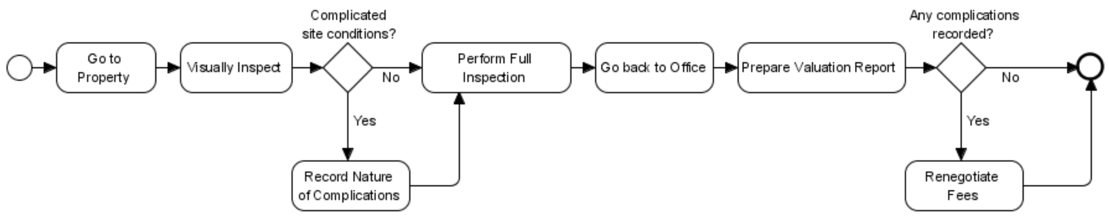

Figure 11: External structure of the 'Perform Valuation' sub-process

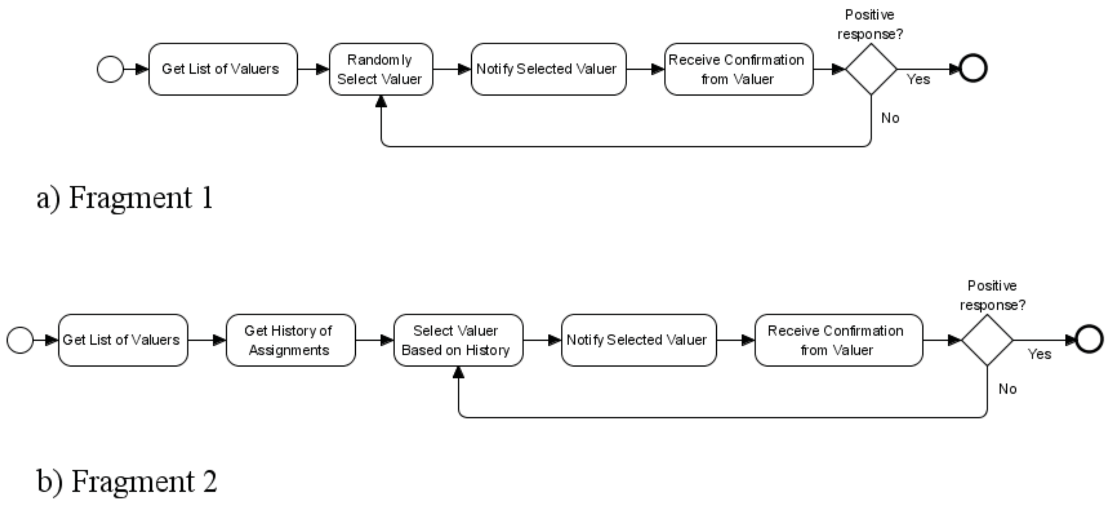

Figure 12: Two 'top-down selection'- type process fragments ((a) Fragment 1 and (b) Fragment 2) can be selected as expected structures of the 'Assign Valuer' sub-process

using a necessary and sufficient set of specifications. The examples in this paper demonstrate a range of process designs that can be generated based on different process specifications.

The genericity of the methodology has the benefit that it can be applied to any process in any domain. It can capture any process view, such as control-flow, data-flow, and organisational views. Although the methodology was illustrated in the paper using annotated BPMN diagrams, it is independent of the use of any particular process modelling notation. On the flip side, the main limitation of this generic methodology is that it provides little domain-specific decision-making support. For example, selecting which parts of a process are to be specified in a robust, scalable or modifiable way remains a challenge for domain experts. And as more experience with the dynamics of a particular process environment becomes available over time, some previously anticipated change effects might turn out unrealistic, which could then require changes in the original process specification. No methodological support is currently available for making these specification decisions.

All the examples presented in this Section are extracted from the ongoing engagement of our research group with the Australian lending industry. As yet, we have not gathered any experiences with industrial applications of using the proposed methodology. However, we are currently developing resource-oriented architecting technologies (Xu et al. 2008) to enable process flexibility in distributed, web-based environments, using the methodological foundations presented in this paper. One of the issues to be addressed is the definition of a domain ontology that is shared among stakeholders. A lightweight

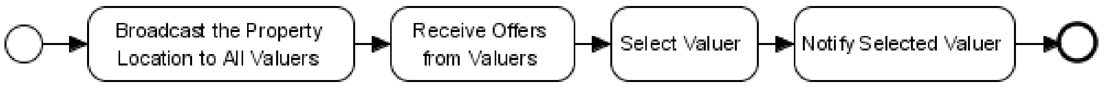
Broadcast the Property Location to All Valuers
Receive Offers from Valuers
Select Valuer
Notify Selected Valuer

Figure 13: A ‘bidding’- type process fragment selected as a new class of expected structure of the ‘Assign Valuer’ sub-process

but sufficiently well defined ontology may enable unambiguous communication of process requirements to human operators. Formally verifiable ontologies and executable process notations are needed for applying the methodology in automated process environments. Progress in this area is likely to draw on research in process patterns (Aalst et al. 2003), exception-handling patterns (Lerner et al. 2010), configurable process models (Gottschalk et al. 2008) and semantic business process management (Wetzstein et al. 2007).

There are opportunities to expand the design view of process flexibility and thus to refine the methodology, using further analogies from engineering design. One research direction may focus on the qualitative relationships between function, behaviour and structure state spaces. Capturing them can guide process designers realising desired changes of function through appropriate changes of behaviour and structure. For example, research in product family design maps different types of product families (that can be modelled as sets of scalable or modifiable behaviours and structures) onto strategies for targeting different market segments (that can be modelled as sets of scalable or modifiable functions) and onto different manufacturing paradigms (that can be modelled as design realisation options) (Maier and Fadel 2007). Research in PAIS is likely to benefit from further investigation of these analogies, as they allow tapping into well established methodologies of flexibility from various domains.

5 Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

van der Aalst W. M. P., Jablonski S. (2000) Dealing with Workflow Change: Identification of Issues and Solutions. In: Computer Systems Science and Engineering 15(5), pp. 267–276

van der Aalst W. M. P., Weijters A. J. M. M. (2004) Process Mining: A Research Agenda. In: Computers in Industry 53(3), pp. 231–244

van der Aalst W. M. P., ter Hofstede A. H. M., Kiepuszewski B., Barros A. P. (2003) Workflow Patterns. In: Distributed and Parallel Databases 14(3), pp. 5–51

Adams M., ter Hofstede A. H. M., Edmond D., van der Aalst W. M. P. (2006) Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: Proceedings the 14th International Conference on Cooperative Information Systems (CoopIS’06). Springer, Berlin, pp. 291–308

van Aken J. E. (2007) Design Science and Organization Development Interventions: Aligning Business and Humanistic Values. In: Journal of Applied Behavioral Science 43(1), pp. 67–88

Borch S. E., Stefansen C. (2006) On Controlled Flexibility. In: Proceedings of Workshops and Doctoral Consortium, The 18th International Conference on Advanced Information Systems Engineering – Trusted Information Systems. Namur University Press, Namur, pp. 121–126

Brown K. (1997) Grammatical Design. In: IEEE Expert: Intelligent Systems and their Applications 12(2), pp. 27–33

Daoudi F., Nurcan S. (2007) A Benchmarking Framework for Methods to Design Flexible Business Processes. In: Software Process: Improvement and Practice 12(1), pp. 51–63

Dietz J. L. G. (2006) Enterprise Ontology: Theory and Methodology. Springer, Berlin

Dietz J. L. G. (2008) On the Nature of Business Rules. In: Advances in Enterprise Engineering I. Springer, Berlin, pp. 1–15

Gero J. S. (1994) Towards a Model of Exploration in Computer-Aided Design. In: Formal Design Methods for CAD. North-Holland, Amsterdam, pp. 315–336

Gero J. S., Fujii H. (2000) A Computational Framework for Concept Formation for a Situated Design Agent. In: Knowledge-Based Systems 13(6), pp. 361–368

Gero J. S., Kannengiesser U. (2004) The Situated Function-Behaviour-Structure Framework. In: Design Studies 25(4), pp. 373–391

Gero J. S., Kannengiesser U. (2007) A Function-Behavior-Structure Ontology of Processes. In: Artificial Intelligence for Engineering Design, Analysis and Manufacturing 21(4), pp. 379–391

Gorti S. R., Gupta A., Kim G. J., Sriram R. D., Wong A. (1998) An Object-Oriented Representation for Product and Design Processes. In: Computer-Aided Design 30(7), pp. 489–501

Gottschalk F., van der Aalst W. M. P., Jansen-Vullers M. H., La Rosa M. (2008) Configurable Workflow Models. In: International Journal of Cooperative Information Systems 17(2), pp. 177–221

Greer J. L., Stock M. E., Stone R. B., Wood K. L. (2003) Enumerating the Component Space: First Steps toward a Design Naming Convention for Mechanical Parts. In: 2003 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Chicago, pp. 1–12

Kannengiesser U. (2008) Subsuming the BPM Life Cycle in an Ontological Framework of Designing. In: Advances in Enterprise Engineering I. Springer, Berlin, pp. 31–45

Kannengiesser U. (2009) Can We Engineer Better Process Models? In: Proceedings of the 17th International Conference on Engineering Design (ICED'09), Vol. 1. The Design Society, pp. 527–538

Kruchten P. (2005) Casting Software Design in the Function-Behavior-Structure Framework. In: IEEE Software 22(2), pp. 52–58

Lapouchnian A., Yu Y., Mylopoulos J. (2007) Requirements-Driven Design and Configuration Management of Business Processes. In: Business Process Management. Springer, Berlin, pp. 246–261

Lerner B. S., Christov S., Osterweil L. J., Bendraou R., Kannengiesser U., Wise A. (2010) Exception Handling Patterns for Process Modeling. In: IEEE Transactions on Software Engineering 36(2), pp. 162–183

Logan B., Smithers T. (1993) Creativity and Design as Exploration. In: Modeling Creativity and Knowledge-Based Creative Design. Lawrence Erlbaum, Hillsdale, pp. 139–175

Maier J. R. A., Fadel G. M. (2007) A Taxonomy and Decision Support for the Design and Manufacture of Types of Product Families. In: Journal of Intelligent Manufacturing 18(1), pp. 31–45

Papalambros P., Wilde D. J. (2000) Principles of Optimal Design: Modeling and Computation. Cambridge University Press, Cambridge

Pesic M., van der Aalst W. M. P. (2006) A Declarative Approach for Flexible Business Processes Management. In: BPM 2006 Workshops. Springer, Berlin, pp. 169–180

Regev G., Soffer P., Schmidt R. (2006) Taxonomy of Flexibility in Business Processes. In: Proceedings of Workshops and Doctoral Consortium, The 18th International Conference on Advanced Information Systems Engineering – Trusted Information Systems. Namur University Press, Namur, pp. 90–93

Regev G., Bider I., Wegmann A. (2007) Defining Business Process Flexibility with the Help of Invariants. In: Software Process: Improvement and Practice 12(1), pp. 65–79

Rinderle S., Reichert M., Dadam P. (2004) Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey. In: Data & Knowledge Engineering 50(1), pp. 9–34

Rosemann M., Recker J., Flender C. (2008) Contextualisation of Business Processes. In: International Journal of Business Process Integration and Management 3(1), pp. 47–60

Ross A. M., Rhodes D. H., Hastings D. E. (2008) Defining Changeability: Reconciling Flexibility, Adaptability, Scalability, Modifiability, and Robustness for Maintaining System Lifecycle Value. In: Systems Engineering 11(3), pp. 246–262

Sadiq S., Sadiq W., Orlowska M. (2001) Pockets of Flexibility in Workflow Specification. In: Conceptual Modeling – ER 2001. Springer, Berlin, pp. 513–526

Saleh J. H., Hastings D. E., Newman D. J. (2003) Flexibility in System Design and Implications for Aerospace Systems. In: Acta Astronautica 53(12), pp. 927–944

Schön D. A. (1983) The Reflective Practitioner: How Professionals Think in Action. Harper Collins, New York

Schonenberg H., Mans R., Russell N., Mulyar N., van der Aalst W. M. P. (2008) Process Flexibility: A Survey of Contemporary Approaches. In: Advances in Enterprise Engineering I. Springer, Berlin, pp. 16–30

Snowdon R. A., Warboys B. C., Greenwood R. M., Holland C. P., Kawalek P. J., Shaw D. R. (2007) On the Architecture and Form of Flexible Process Support. In: Software Process: Improvement and Practice 12(1), pp. 21–34

Soffer P., Regev G. (2005) On the Notion of Soft-Goals in Business Process Modeling. In: Business Process Management Journal 11(6), pp. 663–679

Weber B., Reichert M., Rinderle-Ma S. (2008) Change Patterns and Change Support Features – Enhancing Flexibility in Process-Aware Information Systems. In: Data & Knowledge Engineering 43(1), pp. 67–88

Weber B., Shazia S., Reichert M. (2009) Beyond Rigidity – Dynamic Process Lifecycle Support. In: Computer Science – Research and Development 23(2), pp. 47–65

Weick K. E. (1995) Sensemaking in Organizations. Sage, Reading

Wetzstein B., Ma Z., Filipowska A., Kaczmarek M., Bhiri S., Losada S., Lopez-Cobo J. M., Cicurel L. (2007) Semantic Business Process Management: A Lifecycle Based Requirements Analysis. In: Semantic Business Process and Product Lifecycle Management. Proceedings of the Workshop SBPM 2007. Innsbruck, pp. 1–10

Xu X., Zhu L., Liu Y., Staples M. (2008) Resource-Oriented Architecture for Business Processes. In: 15th Asia-Pacific Software Engineering Conference. Beijing, pp. 395–402

Udo Kannengiesser

NICTA,
Locked Bag 9013,
Alexandria NSW 1435,
Australia,
and
School of Computer Science and Engineering,
University of New South Wales, Sydney,
Australia
udo.kannengiesser@nicta.com.au