Towards Concise Architectures for Flexible Business Processes

Udo Kannengiesser and Liming Zhu

NICTA, Australia, and
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

{udo.kannengiesser, liming.zhu}@nicta.com.au

Abstract

This chapter proposes a view of business processes as designed artefacts that are ontologically no different than artefacts in domains such as mechanical and software engineering. This view distinguishes three concerns for designing processes: architecture, implementation and adaptation. We show that current process modelling approaches conflate these aspects, often leading to high complexity and inflexibility of the resulting process models. We use a generalisation of the “feature” concept in engineering design, represented using the function-behaviour-structure (FBS) ontology, as the basis of a new approach to concisely specifying business process architectures that allow for more process flexibility.

1. Introduction

Business process modelling is an area that deals with creating representations of business processes for various purposes, including business process analysis, understanding, communication, standardisation, simulation, improvement and implementation. Although a number of different, mostly graphical notations and tools for modelling business processes have been developed (Dumas et al. 2005), their effectiveness is often reduced due to a number of issues that remain topics of ongoing research. One of the issues is the high complexity of many process models, which significantly affects understanding of these models by human experts (Bandara et al. 2007). This problem is commonly perceived as a problem of model granularity, to be addressed by striking a balance between comprehensibility and precision.

Another issue is the poor flexibility of most business process models (Regev et al. 2007). 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 may need to be handled flexibly, such as resource bottlenecks or effects of unexpected human or system errors. Process models that are too rigidly defined are poorly applicable in real-world contexts and are ultimately rejected by their users.
In this chapter, we argue that the issues of complexity and flexibility of process models are due to an inadequate understanding of processes as designed artefacts. Drawing analogies with the domains of mechanical and software engineering, we develop a framework that clarifies the distinction between three concerns in process design (Kannengiesser 2009a): architecture, implementation and adaptation. We show that current business process modelling approaches conflate these aspects, often leading to high complexity and inflexibility of the resulting process models. We use a generalisation of the “feature” concept in engineering design, represented using the function-behaviour-structure (FBS) ontology (Gero 1990; Gero and Kannengiesser 2004), as the basis of a new approach to concisely specifying process architectures that allow for more process flexibility. We demonstrate our approach using examples of a property valuation process in the real estate industry.

2. Artefacts

Artefacts can be defined as entities made by humans to achieve a set of objectives. They can be modelled using the function-behaviour-structure (FBS) ontology (Gero 1990; Gero and Kannengiesser 2004) that has been applied to various instances of artefacts, including physical products (Gero and Kannengiesser 2004), software (Kruchten 2005) and processes (Gero and Kannengiesser 2007).

Structure (S) is defined as an artefact’s components and their relationships (“what the artefact consists of”). 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 (Gero 1990). The structure of software consists of more abstract concepts that cannot be perceived directly using human sensory capabilities. It “exists” only as a set of high-level constructs, represented graphically or symbolically. In the domain of processes, structure includes three general classes of interrelated components: input, transformation and output (Gero and Kannengiesser 2007). Here, the transformation often consists of a coherent set of sub-transformations (or sub-processes). Similar to software structure, most of the individual components of process structure are fairly abstract entities that are often described using boxes and arrows.

Behaviour (B) is defined as the attributes that can be derived from an artefact’s structure (“what the artefact does”). 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 (e.g., operating) environment. Typical behaviours of processes include speed, cost, precision and accuracy. They can also be derived from structure properties; for example, speed can be derived from time stamps on input and output.

Function (F) is defined as an artefact’s teleology (“what the artefact is for”). This notion is independent of the common distinction between “functional” and “non-functional” properties; it comprises both as long as they describe the artefact’s usefulness for a stakeholder.
Function is ascribed to behaviour by establishing a teleological connection between a human’s goals and measurable effects of the artefact. The particular functions of an artefact are ontologically independent of whether the artefact’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 set of tasks (i.e., a process).

Artefacts are realised by processes that transform a set of “raw materials” (input) into final artefacts (output). This is in contrast to the widely held view of realisation as transforming artefact models into “real” artefacts. While it is true that realisation is guided by artefact models and other (represented) instructions, what this process ultimately transforms is only the “real” world (or “target” world – the world in which the artefact is intended to be used). Table 1 presents three specific realisation processes from the domains of mechanical engineering, software engineering and business processes. It shows that business processes are realised by transforming the potential capabilities of individual people, organisations or services (as “raw materials”) into a coherent set of “actual” activities that compose the business process.

One of the specificities of processes, not shown in Table 1, is that realisation time is almost identical with use time. This means that the (use-related) functions of a realised instance of the process artefact come into effect during, or upon completion of the final step of, its realisation. This is different in the domains of mechanical and software engineering, where the use time of artefacts commences well after their realisation time, and not until a number of post-realisation activities, such as packaging, delivering and installing, are completed. We will return to this difference later.

<table>
<thead>
<tr>
<th>Common name</th>
<th>Mechanical engineering</th>
<th>Software engineering</th>
<th>Business processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embodiment</td>
<td>Physical world</td>
<td>Computational world</td>
<td>Business world</td>
</tr>
<tr>
<td>Input</td>
<td>Physical materials and components</td>
<td>Computing platform</td>
<td>Capabilities of people, organisations, services</td>
</tr>
<tr>
<td>Output</td>
<td>Physical products</td>
<td>Software products</td>
<td>Business processes</td>
</tr>
<tr>
<td>Transformer</td>
<td>People and machines (e.g., drills, lathes, assembly robots)</td>
<td>Virtual machines, interpreters</td>
<td>People, organisations, services</td>
</tr>
</tbody>
</table>

3. Designing

Designing comprises activities that aim at producing representations of artefacts given a set of requirements. We propose three distinct concerns in designing (Kannengiesser 2009a): architecture, implementation and adaptation.
3.1. Architecture

The notion of architecture in traditional design domains such as building design and (physical) product design (Ulrich 1995) has long been used to inspire approaches to defining architecture in software engineering. Here, architecture is commonly understood as comprising a set of components, their relationships, and the rationale oriented to various functional and non-functional requirements and constraints (Perry and Wolf 1992). Central in the representation of software architecture is the use of multiple views to allow for various forms of analysis (Kruchten 1995; Bass et al. 2003). Process architecture has been defined in similar ways. Here, the components and their relationships usually represent interconnected activities and deliverables (Browning 2009), which is consistent with the notion of structure in the FBS ontology. In the business process domain, the notion of process architecture is often understood to comprise the fundamental components that are “essential” for the operations of a specific organisation (Ould 1997; Dietz 2006). Various views have been proposed to account for different process requirements and analysis goals (Curtis et al. 1992; Scheer 2000; Browning 2009).

There are two principal motivations for specifying architectures. One motivation is to obtain high-level, descriptive models of artefacts that allow for easy comprehension and reuse of essential design concepts. The other motivation is to provide constraints for managing changes or variations across different use or realisation environments. The latter is the basis for generating multiple instances of artefacts based on the same fundamental architecture (Jiao and Tseng 2000; Bosch 2000).

3.2. Implementation

Implementation is a process that transforms an architecture (input) into a prescriptive model of the realisation process (output). This model includes very detailed descriptions of the realisation steps, and is presented in a form that can be understood by the realisation transformer (see Table 1). Table 2 shows three specific implementation processes from the domains of mechanical engineering, software engineering and business processes. They are all embodied in a represented world (which may be computational, paper-based or mental) rather than in the target world.

<table>
<thead>
<tr>
<th>Table 2. Three domain-specific views of implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common name</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Embodiment</td>
</tr>
<tr>
<td>Input</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Transformer</td>
</tr>
</tbody>
</table>
The output of implementation, in this chapter called the “realisation model”, may be more or less similar to its input (i.e., the architecture) depending on the specific domain. In mechanical engineering, assembly and manufacturing plans are procedural (or process-based) and thus very different from corresponding product architectures that are object-based. (This is despite the fact that the basis of assembly plans is established by mapping every product component onto an individual “assembly step”.) In software engineering, the way in which source code is structured is often very similar to the corresponding software architecture. For example, classes in object-oriented source code often map onto components in the software architecture. In process domains including business processes, there is a very high similarity between business process architectures and their corresponding workflow models. This is because both are process-based, and most activities in the process architecture can be mapped onto activities in the realisation model.

Implementation is not a routine, mechanistic process but an instance of designing, with the realisation process (represented in the realisation model) being the artefact. The implementer needs to create the realisation model in consideration of a variety of additional requirements and constraints related to the realisation environment. Some of these requirements and constraints may not be explicitly specified in the architecture but are constructed from the implementer’s experience and interactions.

3.3. Adaptation

Adaptation refers to the activities needed for modifying the structure of a process, to respond to changed requirements or constraints. The structure to be adapted may be of the process architecture or of the realisation of that process architecture. An important class of changes that require adaptation are related to unexpected events that may occur in the realisation environment. Adaptation is then concerned with recovering instances of the realisation process to mitigate possible undesired effects. This can be done by adding or removing realisation activities, re-sequencing activities, or re-allocating resources (agents) to activities. Adaptation can be viewed as an instance of re-designing.

Take a manufacturing process of mechanical products; this process often needs to be adapted in response to variations in product demand, cost constraints, required capacity utilisation or unexpected machine breakdowns. This results in a re-designed structure of the manufacturing process, by modifying the possible kinds of manufacturing steps, their order, and the allocation of specific machines. In software engineering, source code often needs to be adapted due to unexpected faults, security threats or underperforming resources. This leads to a re-designed realisation structure in terms of modified source code fragments, e.g. by introducing new exception handlers, modifying access control, or reconfiguring interactions between program components. In process domains including business processes, workflows need to be adapted to unexpected events and new constraints, such as order cancellations, service interruptions and changed business rules (e.g., introduce fast-tracked handling of complaints from gold customers). The resulting changes in the workflow may include the addition of activities (e.g., cancel shipment) and the re-allocation of resources (e.g., alternative service, and higher-rank officer in complaint department).
Changes may be temporary, affecting only one or very few instances, or persistent with longer-term effects on the realisation process. No matter what timeframe and scope, all changes lead to re-represented workflow models, which are either mental models formed by the agents performing the realisation process (in which case the changes are viewed as runtime or *ad hoc* changes), or formal “master” models from the implementers (in which case the changes are viewed as design-time or “planned” changes). Agents performing the adaptation are either human or computational. If the agent is computational, the adaptation is commonly termed “self-adaptation”.

### 4. Issues in Current Process Models

The three concerns presented in this chapter have not been well understood in business process modelling. One reason for this is the identical, procedural style of representing architecture, realisation models and adaptation. Their components are conceptualised as activities that all look the same (see Section 3.2). Another reason is the non-persistent nature of process artefacts, and their overlapping realisation and use times (see Section 2). Architecting, which focuses on processes at use time, and implementation, which focuses on processes at realisation time, thus tend to produce identical models even though they are to describe different processes.

This Section presents how issues of complexity and flexibility can arise from an insufficient understanding of the notions of business process architecture, implementation and adaptation. As an example, we use a simplified model of a property valuation process in the Australian mortgage industry, represented as a BPMN (Business Process Modeling Notation) diagram in Figure 1. Here, the property valuation process (short: valuation process) starts when the valuation company receives a valuation request from a lender (e.g., a bank). A specific person (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 1. Property valuation process](image)

This process example is quite simple; however, it is easy to see how the issues to be outlined in this Section would scale up for more detailed business process models.
4.1. Complexity

Processes are often conceptualised as paths (or sets of interrelated paths based on specified conditions) along the dimension of time. Different steps within a path are activated at different times, and their activations “flow” down the specified network of paths. This view is reflected in the term “work-flow”, and is consistent with the prescriptive, instructional stance inherent in realisation models. On the other hand, it is recognised that the design of processes is driven by various goals, ranging from business or organisational goals to technological constraints and quality concerns. All these aspects map onto the notions of function and behaviour that are reflected in process architectures. However, most business process specification languages, such as BPMN, EPC (Event-Driven Process Chain) and YAWL (Yet Another Workflow Language), capture only the structure aspect of processes, using (mostly graph-based) constructs to represent flows and activities. Few approaches propose representations of process goals in addition to process structure (Lapouchnian et al. 2007).

The strong similarity between process architectures and realisation models has led to their conflation and the common practice of merging them in a single process model. This may be convenient in some cases, but can cause a number of issues, one of which is complexity. We use the number of components (nodes) and their relationships (arcs) in a process model as a measure for complexity, as they have been shown to be the predominant model characteristics affecting comprehensibility (Mendling et al. 2007). A frequent cause for increased complexity is the inclusion of the fine-grained details of realisation models in process architectures. For example, activities of communication may be necessary for coordinating the people, departments or services that carry out the realisation process. However, these activities are typically not relevant on the architectural level. This has the effect that the essential, value-adding activities that form the process architecture are obscured by a multitude of ancillary realisation activities.

The complexity of many process models and their concomitant poor comprehensibility is a well-known issue in business process modelling practice. However, this issue is often perceived as a visualisation problem (Bandara et al. 2007) rather than a methodological problem. Currently, the main approach to reducing complexity is the use of hierarchical process structures that chunk some of the detailed activities into sub-processes. This is essentially an information-hiding approach. The problem here is that the complete set of sub-process attributes are hidden, including those that are needed for understanding the distinguishing nature of the specific realisation model.

For example, the “Assign Valuer” and “Perform Valuation” activities in Figure 1 are represented as “collapsed” sub-processes (indicated by the “plus” sign in the lower centre of their shapes). Figure 2 expands the “Assign Valuer” sub-process in a separate diagram, now showing all its finer-grained (realisation) details. What are the fundamental characteristics of this realisation process that distinguish them from alternative ways of realising “Assign Valuer”? One such alternative way is shown in Figure 3. Here, a bidding-style mechanism is introduced to quickly identify those potential valuers that are currently located near the property to be valuated. This mechanism is inspired by the way taxi companies dynamically assign their taxis to specific jobs. The essential differences between the alternative processes in Figures 2 and 3 can be viewed
respectively as a slower, “top-down” versus a faster, self-organising realisation strategy. This higher-level, semantic characterisation is lost in simple information hiding.

4.2. Flexibility

Over-specifying realisation models as sub-processes of process architectures has the additional drawback that there is no room for local variations based on the specific constraints imposed by the realisation environment. This is because architecture, implementation and adaptation are not treated as separate activities that address separate design concerns. Once process architectures are specified that subsume all the details of the associated realisation models, there is almost no flexibility remaining. Take the activity “Assign Valuer” as an example; here, current process modelling notations such as BPMN require a particular sub-process to be defined and associated with this activity. No decision support is provided for this modelling task.

While this issue may be solved by providing meta-data, in many cases dynamic variations are required that need to be based on information captured in the process model itself. These dynamic variations are commonly known as exception handling. Here, all possible situations and events that may interfere with the process are anticipated, and ways are defined in which the process may best deal with these events. A typical example of exception handling in processes is the occurrence of an error within an activity (e.g., a system failure) or some other abnormal situation, and the definition of an additional path within the process that handles that error or abnormal situation (e.g., by repeating the process step, or by allocating the step to a different performer).

Generally, there are no reasons against using this approach. However, in many process domains it is hard if not impossible to reliably predict all possible exceptions that may occur and pre-define appropriate exception-handling strategies. In addition, the “firing” of exceptions depends on whether or not they are actively monitored for and with what techniques of sensing and analysis. These are concerns of process adaptation rather than process architecture.
Figure 4 shows an example of exception handling within the sub-process “Perform Valuation”. It handles cases in which a particular valuation turns out to be difficult due to complicated site conditions, such as irregular building shapes or slopes. The valuer can generally recognise this situation upon initial visual inspection of the site. The exception is then handled by renegotiating (higher) fees with the lender. According to the exception-handling path represented in Figure 4, the renegotiation requires the valuer to go back and forth between office and property, before resuming the normal flow of activities, including performing the full inspection and preparing the valuation report.

![Figure 4. “Perform Valuation” sub-process option 1](image)

However, this is not the only way in which valuers can handle the same exception. Figure 5 shows that fee renegotiation can also be performed at a later stage within the process. This has the advantage that the valuer does not have to do the additional return trip between office and property (as in Figure 4), which can significantly speed up the valuation.

![Figure 5. “Perform Valuation” sub-process option 2](image)

Which of the two exception-handling strategies are selected must be specified a priori. This restricts the valuers’ freedom to flexibly decide which of the alternatives is more appropriate in the specific situation. It is certainly possible to combine both exception-handling paths into one model, Figure 6. However, the usefulness of such a combination is questionable, as the conditions for each path cannot be pre-defined in a sufficiently precise and complete way. Moreover, it is often cumbersome to represent multiple, intertwined paths using traditional flowchart-based notations.
The explicit representation of exception-handling, and associated mechanisms for analysing and evaluating the effect of events on the current process instance, subsumes adaptation activities in the realisation model. In other words, activities of (re-) designing are merged with the artefact. This not only adds to complexity but also makes the process inflexible by limiting the range of possible process changes to those that have been anticipated and pre-defined.

5. Towards a Feature-Based Approach

In this Section, we will present our approach to specifying concise architectures for flexible business processes, based on the “feature” concept in engineering design (Salomons et al. 1993; Shah and Mäntylä 1995). This concept is more general than the software engineering notion of features (as pieces of functionality that are of value to the customer). Features in engineering design describe any portion of an artefact’s FBS representation that is significant in a particular life-cycle context (Brown 2004). For example, materials in mechanical engineering design can be viewed as features, as they are typically specified in terms of those of their FBS properties that are significant for manufacturing and use. Take this material feature labelled according to the DIN EN 10027 norm:

\[ \text{G-S275} \]

This label represents cast structural steel (“G” stands for “cast”; “S” stands for “steel”) with yield strength of 275 N/mm\(^2\). It can be interpreted as an explicit specification of the function, behaviour and structure of material:

- Function: “provide the input for casting processes”
- Behaviour: yield strength \(\geq 275\) N/mm\(^2\)
- Structure: structural type = steel

This example demonstrates two roles of features:

1. Features as high-level building blocks: The FBS description provides high-level, semantic information rather than a low-level description of the material’s molecular structure. Structure is referred to only through a label denoting its type (“steel”), based on its commonly known definition as a ferrous alloy with carbon content of less than 2.06%.
Semantics is added by function and behaviour, as they provide the information relevant to product designers and product manufacturers using the material. The benefit of this representation is that it conveys rich information using a shorthand label.

2. Features as design constraints: The feature provides a set of constraints for the selection of material from a materials database. The feature representation allows specifying only those aspects that are necessary and sufficient for the task at hand. Materials can be selected (or designed) with any attribute values or additional attributes as long as the specified constraints are satisfied. If needed, the conventions in DIN EN 10027 allow increasing the set of constraints in a systematic, standardised way. For example, “G-S275JR” denotes the same material but with the additional behaviour constraint of notched impact strength of 27 Joule at 20 °C (“J” stands for 27 Joule, “R” stands for 20 °C).

Features have the potential to provide a generic tool for specifying artefacts with reduced complexity, through their role as high-level building blocks, and increased flexibility, through their role as design constraints for designing and adaptation. The domain-independent representation based on the FBS schema allows applying this tool to any class of artefact, including business processes. Unfortunately, most business process domains do not have well-established conventions and notations for labelling the FBS properties of processes or activities. However, we can demonstrate the feasibility and usefulness of integrating the feature concept in process architectures using our valuation example. Figure 7 shows that a feature-based architecture of this process can be generated by replacing fine-grained models of realisation details with annotations of their relevant function, behaviour and (high-level) structure. In particular, the Figure shows example specifications for two features, associated with “Assign Valuer” (Feature 1) and “Perform Valuation” (Feature 2).

![Feature-based architecture of a property valuation process](image)

Before looking more closely at the two features, note that we have chosen to represent “Perform Valuation” no longer as a sub-process (as in Figure 1) but as a loosely connected group of two core activities. This is not because we think sub-processes were generally not useful, but to emphasise that there is no one-to-one mapping between the components of process architectures.
and the components of their realisation. They may happen to have identical names, but they belong to fundamentally different design concerns.

The features in Figure 7 can be seen as extensions of the sub-processes in Figures 2 to 6 that are specific instances of the realisation of the valuation process. This is because the features describe relevant FBS portions of realisation that are no longer limited to low-level details of a fixed realisation structure. For instance, the two (mutually exclusive) examples for specifying Feature 1 capture the essence of the different mechanisms for realising “Assign Valuer”, in terms of their different high-level structure (“top-down” in Example 1 versus “bidding” in Example 2) and behaviour (time within 1 business day in Example 1 versus within 1 hour in Example 2). This provides a concise way of representing the relevant characteristics of the required realisation options. It also allows for more flexibility, as it makes no commitment to specific realisation details. This is in contrast to the sub-processes in Figures 2 and 3, where every activity needs to be specified explicitly.

Examples 3 and 4 for specifying Feature 2 capture alternative ways in which “Perform Valuation” can be realised. Example 3 provides a specification of the behaviour but not the structure of realisation. Here, the feature may cover any realisation structure (such as those defined in Figures 4 to 6) as long as that structure satisfies the specified behaviour. Example 4 provides a more constrained feature description, with a reduced range of possible behaviour values and a high-level specification of the mechanism required for adapting the realisation process. In this example, the adaptation mechanism is specified as “deferred fixing” that is one of the exception-handling patterns proposed by Lerner et al. (2010). The essential idea behind this pattern is that an exceptional situation is recognised and recorded, but dealt with later in the process. This fits with the sub-process in Figure 5 that records the nature of the complication but handles the consequences (i.e., the higher fees) later, after completing the full property inspection. However, this high-level pattern still allows for variations on a more detailed level. For example, one may choose to realise the activity “Renegotiate Fees” not before (as shown in Figure 5) but after preparing the valuation report.

For simplicity, our feature examples include not more than one property or constraint per ontological category. It is possible to expand the FBS descriptions as needed; for example, by adding quality attributes such as compliance, robustness and cost. As any number of constraints may be included, the reduction in process model complexity achieved through features needs to be offset against potential increases in feature complexity. However, considering that even a small number of low-level process variations can lead to highly complex process models (as illustrated in Figure 6), the use of features is very likely to reduce overall model complexity by abstracting away from structure details. Future research may investigate approaches based on distinct architectural views (Kruchten 1995; Browning 2009) to be used as information filters for cases where feature complexity becomes an issue.

We can quantify the reduction in complexity for our example. Table 3 compares the traditional and the feature-based approach based on the number of nodes and arcs of models of the valuation process using different combinations of realisation options. As there are two options presented for realising the “Assign Valuer” activity (Figures 2 and 3) and three options presented for
realising the “Perform Valuation” activity (Figures 4, 5 and 6), we consider six combinations of options. We refer to every combination of options as a realisation instance, described as a tuple (AssignVal, PerformVal), with AssignVal and PerformVal representing specific realisation options for “Assign Valuer” and “Perform Valuation”, respectively. For example, the tuple (2, 1) represents the realisation instance that uses option 2 for “Assign Valuer” (Figure 3), and option 1 for “Perform Valuation” (Figure 4). The Table shows that the traditional approach requires higher numbers of nodes and arcs than the feature-based approach. This is because it includes the nodes and arcs of all the realisation details, captured as sub-processes. In contrast, the feature-based approach extracts the essence of these details and adds it as annotations to the process architecture. The resulting reduction in complexity is significant, averaging more than 50% with respect to both nodes and arcs. It can be expected that the feature-based approach can lead to even higher reductions of complexity for larger business processes.

Table 3. Complexity of traditional and feature-based models of the valuation process

<table>
<thead>
<tr>
<th>Realisation instance</th>
<th>Complexity when using traditional approach</th>
<th>Complexity when using feature-based approach</th>
<th>Reduction in complexity when using feature-based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of nodes</td>
<td>No. of arcs</td>
<td>No. of nodes</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>28</td>
<td>27</td>
<td>13</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>28</td>
<td>28</td>
<td>13</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>32</td>
<td>33</td>
<td>13</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>27</td>
<td>25</td>
<td>13</td>
</tr>
<tr>
<td>(2, 2)</td>
<td>27</td>
<td>26</td>
<td>13</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>31</td>
<td>31</td>
<td>13</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>28.8</strong></td>
<td><strong>28.3</strong></td>
<td><strong>13</strong></td>
</tr>
</tbody>
</table>

The flexibility of the feature-based approach is difficult to evaluate quantitatively. However, it can be shown qualitatively that it introduces a new class of flexibility that can be called “design freedom”. It is well demonstrated by Example 3 in Figure 7, which allows for maximum design freedom by specifying requirements pertaining only to the function and behaviour of “Perform Valuation” and not its structure. Existing approaches to process flexibility have been shown to be based on specifying constraints only on structure but not behaviour and function (Kannengiesser 2009b).

6. Related Work

The notions of architecture, implementation and adaptation, in the way presented in this chapter, have not previously been applied to processes. Yet, there has been some work related to the issues of complexity and flexibility in business process modelling.
A number of approaches to reducing process complexity are based on transformations of given process models. For example, graph reduction rules have been used for control-flow verification (Sadiq and Orlowska 1999). An approach by Polyvyanyy et al. (2008) identifies less significant activities in a process model based on measures of probability and effort. A new process model is then generated by either eliminating these activities or aggregating them with other activities. However, these approaches reduce model complexity without extracting and preserving higher-level semantics. This results in a potential loss of vital information for understanding, analysing and evaluating the process. Other approaches are limited to process model syntax. For example, the design structure matrix has been used as a compact schema for visualising and analysing interdependencies among activities (Browning 2002).

Process patterns (Van der Aalst et al. 2003) and process reference models (Motschnig-Pitrik et al. 2002) provide generic process descriptions that may be suited for a common ontology of higher-level realisation structures. We have demonstrated this use of an (exception-handling) pattern as part of our feature-based approach in Section 5. However, most existing patterns and reference models provide little support for reuse (Reinhartz-Berger et al. 2005).

The issue of process flexibility has been addressed by work on state space representations of processes (Regev et al. 2007). Flexibility can be understood here as the ability to move within a state space by selecting different values of the states within given ranges. Different instantiations of this concept have been proposed, including process constraints and process fragments (Sadiq et al. 2005), and parameters for individual activities within a process (Simidchieva et al. 2007). Business process modelling experts may then specify only a “core process”, allowing for the late binding of values to invariants according to individual or dynamically emerging needs. However, what is missing in these approaches is the explicit consideration of goals and requirements. Their focus has thus far been on setting up and constraining the state space of process structure but not process behaviour or process function. As a result, there is no way of specifying criteria that guide the selection of appropriate process structures.

7. Conclusion

The contributions of this chapter are twofold. First, the chapter provides a domain-independent understanding of designing and the concerns of architecture, implementation and adaptation. This understanding sheds new light on the known issues of complexity and flexibility in business process modelling. They now appear as symptoms of the more fundamental problem of separating the three design concerns.

Second, the chapter proposes the feature concept in engineering design as the basis of a general technique for specifying the relationships between the different concerns. The FBS ontology is used for systematically and uniformly representing all features. This allows specifying concise business process architectures that constrain implementation and adaptation to an extent that is necessary and sufficient for the particular goals and requirements imposed by different business process environments. Business processes that are realised based on feature-based architectures are more flexible than those based on conventional business process models. This is because
business process re-designing (i.e., adaptation) is no longer in the straightjacket of a fixed process model but delegated to local, dynamic decision-making controlled by design constraints.

Our approach has the potential to drive further research in both business process modelling and software architecture. In business process modelling, we see our work as a motivator for new efforts in describing process patterns and reference models directed towards better reuse as high-level building blocks for process design. Domain ontologies need to be developed to define standards for systematic and extensible labelling of business processes that reflect their function, behaviour and structure.

Our re-conceptualisation of business process modelling also provides opportunities for developing new software engineering approaches to supporting the design, analysis and enactment of services and business processes. Specifically, software architectures are to be tailored to the different needs of the three process design concerns. A suitable basis for developing these architectures may be the FBS framework that represents all instances of designing uniformly, including architecting, implementing and adapting business processes.

Acknowledgments

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


