

# Achieving Flexibility with Object-Oriented Process Models

Guy Redding<sup>1</sup>, Marlon Dumas<sup>2,1</sup>, Arthur H.M. ter Hofstede<sup>1</sup> and Adrian Iordachescu<sup>3</sup>

<sup>1</sup> Queensland University of Technology, Australia  
{g.redding, a.terhofstede}@qut.edu.au

<sup>2</sup> University of Tartu, Estonia  
marlon.dumas@ut.ee

<sup>3</sup> FlowConnect Pty Ltd, Australia  
adrian@sws.com.au

**Abstract.** Mainstream business process modelling techniques promote a design paradigm wherein the activities that may be performed within a case, together with their usual execution order, form the backbone on top of which other aspects are anchored. This Fordist paradigm, while effective in standardised and production-oriented domains, breaks when confronted with processes in which case-by-case variations and exceptions are the norm. We contend that the effective design of flexible processes calls for a substantially different modelling paradigm. Motivated by requirements from the human services domain, we explore the hypothesis that a framework consisting of a small set of coordination concepts, combined with established object-oriented modelling principles, provides a suitable foundation for designing highly flexible processes. Several human service delivery processes have been designed using this framework and the resulting models have been used to realise a system to support these processes in a pilot environment. This paper presents the framework and illustrates how it addresses different types of flexibility requirements.  
*Keywords:* object-oriented model, process model, flexible process

## 1 Introduction

Process-Aware Information Systems, such as traditional Workflow Management Systems, have difficulties supporting dynamic business processes because they rely on modelling paradigms that tend to impose a given execution order between activities and decision points. This fact has been discussed in the literature for some time leading to many proposals for *flexible workflow* support [1–4].

In this paper, we show how highly flexible business processes can be captured using an object-oriented approach. The proposed approach is inspired by, but arguably not limited to, the delivery of human and social services. Modelling and executing processes in this domain tends to be challenging when compared to other more standardised domains such as insurance and banking. A key feature of delivering human and social services is that the type, number and order of tasks

and sub-processes needed to address a case are often not known until runtime. Also, in these processes, variations on a case-by-case basis and exceptions are the norm. An attempt to impose a standard way of delivering social services is usually met with resistance by the stakeholders involved in the process – both from the providers and consumers of social services.

As a motivating scenario, we consider a process executed in the context of a charity organisation.<sup>4</sup> A recently homeless family contacts a charity and makes an application for assistance. The charity opens a case to manage the family’s homelessness issue. During management of the homelessness case it is discovered that there are additional alcoholism and gambling issues that individual family members require assistance with. Each of these issues can be mapped to a social service that are offered by the charity, but the actual delivery of these services remains *unplanned*. An unplanned situation is particularly challenging to capture using traditional process modelling notations due to the possibility that several potential execution scenarios for a single process model must be captured at design-time. A system that can coordinate unplanned situations requires a framework which supports several types of flexibility but can also enforce constraints where necessary.

In this paper, we explore the hypothesis that an object-oriented modelling approach – based on the principles of data and behaviour encapsulation, event-based communication and polymorphism – provides a suitable basis for capturing the extreme levels of process flexibility needed to manage human social services. The main contribution is a meta-model for the design of highly flexible processes based on object-oriented concepts. The meta-model has been embodied in a modelling tool that allows us to design process models and to export these models in a format suitable for simulation and analysis.

The paper is organised as follows. In Section 2 we formulate flexibility requirements that are addressed by the proposal. Section 3 introduces the elements of the framework. Section 4 demonstrates how these elements can address the flexibility requirements. Section 5 presents related literature and the conclusions drawn from this work are in Section 6.

## 2 Patterns of Flexibility

In our experience in applying Object-Oriented (OO) approaches to design process-aware systems that need to deal with ad-hoc situations, a range of requirements have been observed that we have condensed into three *patterns of flexibility*. A pattern of flexibility is a recurrent problem wherein a designer needs to account for the fact that a variety of circumstances could be encountered during the execution of a process model, yet the scope of these circumstances needs to be captured at design-time to achieve some uniformity (since an organisation provides a finite number of services) or to enforce certain constraints. Each pattern of flexibility also involves a class of users (e.g. social workers or case managers). For convenience we refer to these patterns of flexibility as PoF1-PoF3.

<sup>4</sup> This study is inspired by a project in which one of the co-authors is involved.

**PoF1: Creation Flexibility**

Creation flexibility is the ability of a user to trigger the creation of one or more task instances in an unplanned manner during execution of a process. This pattern of flexibility is needed when the type and number of tasks that will be instantiated as well as the ordering in which these instantiations occur is not known at design-time. Creation flexibility is similar to the case handling approach [5] where tasks do not need to be performed in a strict order and do not necessarily have to be completed to complete a case. At the same time, it is necessary to define constraints regarding the number of task instances and/or the state(s) in a process where unplanned task instances can be created.

Generally speaking, a task instance is created in either a *planned* or an *unplanned* manner. A *planned* task is created according to the control-flow logic of a process model. An *unplanned* task is created *on-demand*, i.e. if and when the task is required. For example, a *Health Assessment* task may require additional tasks that correspond to types of *Treatment*, but for a *Health Assessment* the exact treatments may be difficult to plan at design-time because a particular treatment depends on the result of the assessment.

**PoF2: Delegation Flexibility**

Delegation flexibility is the ability of a user to trigger the transfer of context and data from an executing task to a different task. This pattern of flexibility allows a system to handle situations that begin with poor information concerning an under-specified issue (i.e. assist the client with their homelessness issue and any other issues found). Due to circumstances that frequently affect the delivery of human social services, situations regularly occur that require the context and data from a task to be fully transferred to another (specialist) task.

Such situations can be supported if we allow a new task (called the *delegatee*) to take over the execution of a previous task (the *delegator*). For the purposes of control-flow, the delegatee replaces the delegator, meaning that when the delegatee completes, the completion is treated as if the delegator had completed. This feature, together with the fact that data is fully transferred from the delegator to the delegatee, distinguishes delegation flexibility from creation flexibility. The delegation relation is transitive, meaning that a delegatee may also transfer its execution to another task. Note that from a data-flow perspective, the delegatee is a subtype of the delegator, since the delegatee needs to receive as input the data collected by the delegator and to produce as output at least the same data as the delegator.

**PoF3: Nesting Flexibility**

Nesting flexibility is the ability of a user to create instances of nested sub-processes as they are needed. For example, during the execution of a homelessness process a social worker may discover an additional major issue with the client concerning an alcoholism issue which is well beyond the scope of the

original process that manages homelessness issues. Similar to (task) creation flexibility, nesting flexibility is sometimes only allowed under certain constraints (e.g. the number of sub-processes can be bounded or unbounded and the type of sub-processes can only be created in designated states of a process). However, nesting flexibility deals with creating sub-processes rather than creating tasks – a situation that we call a *referral*. This pattern of flexibility enables a system to create as many layers of ad-hoc sub-processes as needed to manage issues as they arise at runtime, while most importantly retaining process control.

In Section 3 we introduce an OO design framework that enables us to model these patterns of flexibility.

### 3 Flexible OO Processes

We aim to fulfill the objective of achieving flexibility in object-oriented models by proposing a design framework consisting of three abstract types of business objects, namely the Coordination Object, Task Object and Referral Object. These objects are used to construct process models that can capture the patterns of flexibility (PoF1-PoF3) introduced in the previous section. In this section we describe the properties and interactions of these objects.

A **Coordination Object** (COROB) is an object which *coordinates* a process. The COROB is inspired by the recognition that a clear separation must be made between the tasks managed by a process and how the tasks are connected. The net outcome is known as coordination, which explains how this object gets its name. A COROB is responsible for both the creation and synchronisation of the tasks needed to complete a process, managing the overall execution of a process as well as referring out of scope work to other COROBs.

A **Task Object** (TOB) is an object that represents a task. A TOB manages task execution and reports task completion to its parent object. For example, two TOBs in the social services process model are the *Report Collection* and *Client Visit* which both have the *Client Intake COROB* as their parent.

A **Referral Object** (ROB) is an object that allows a COROB to refer a situation which is outside of its scope to another COROB. For example, if several unplanned major issues appear during the execution of a *Homelessness COROB* such as an *Alcoholism* or *Drug Dependency* issue, a ROB is created that operates under the guidance of a user to create the necessary COROB instance.

An *object-oriented process model* consists of a set of object types (COROB, TOB and ROB subtypes) and their relations. A meta-model of the object types and their relations is shown in Figure 1 as a UML Class Diagram. An object lifecycle is captured as a finite state machine which consists of states and transitions. Each state machine has one initial state and one final state. A transition may have an optional event, condition and/or timeout. A gateway is a sub-state of a state that can send and/or receive signals either at the pre-gateway (before a state is entered) or at the post-gateway (after a state is exited). Every state has a pre-gateway and a post-gateway. A creation region is a collection of one or more states in a state machine from within which it is possible to create ob-



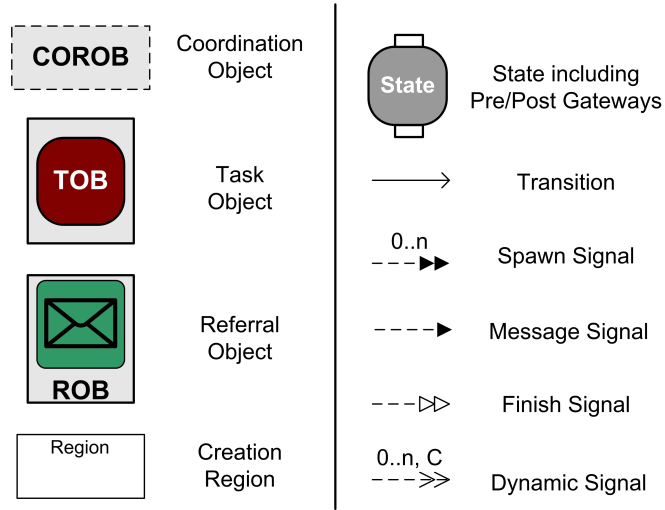


Fig. 2. Framework Elements

In contrast, a dynamic signal allows a process designer to model object communications that *may* occur, meaning that users have the possibility of triggering a dynamic signal, but they may or may not do so. The source of a dynamic signal is a creation region and the target is an object type. If the state of a source object is within the creation region, users are offered the possibility to trigger the dynamic signal. When the dynamic signal is triggered, an instance of the target object type (or one of its subtypes) is created. The target object type depends on a selection strategy associated to the dynamic signal and input given by the user when triggering the dynamic signal. This approach follows the principle of the Strategy Pattern [6].

There are four dynamic signal subtypes: the delegation, creation, referral and nesting signal. A *delegation signal* allows delegation from a creation region within a source delegator TOB to a target delegatee TOB. A delegator may delegate to more than one type of delegatee, which must be a subtype of the delegator. A *creation signal* enables instances of a TOB to be created from a creation region. The difference between delegation and creation signals is the following. When a delegation signal is triggered, the source object ceases to exist and is replaced by the target object. Meanwhile, in the case of a creation signal, a new target object is created and the source object continues to exist. A parent-child relationship is then established between the source object and the newly created object.

Creation and delegation signals serve to transfer control to a TOB. On the other hand, referral and nesting signals serve to transfer control to a COROB. A user may trigger a *referral signal* if an issue arises during the execution of a COROB that falls outside the scope of the COROB. The newly created ROB then assists users in finding a suitable COROB type to address the issue in question. During the execution of a ROB, a user (not necessarily the same who

created the ROB) may then trigger a *nesting signal*, resulting in the creation of a new COROB to handle the issue in question.

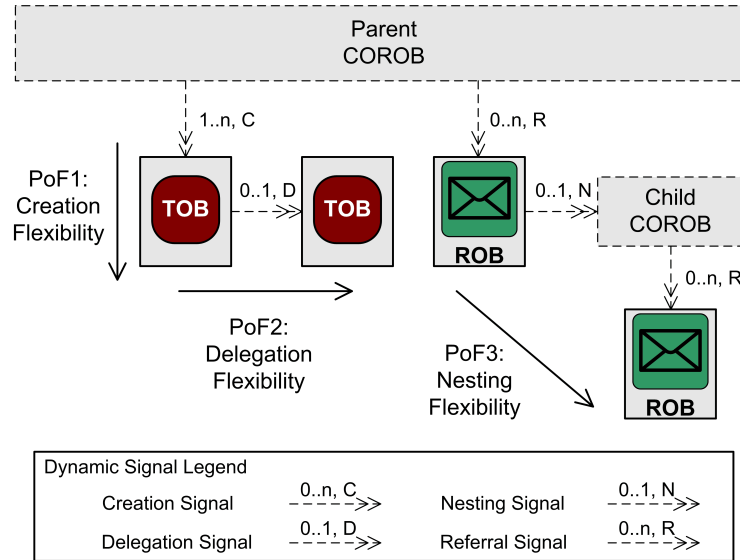


Fig. 3. Patterns of Flexibility in the Framework

Figure 3 outlines how the three object types and the dynamic signals in the framework are structured to capture each pattern of flexibility. In Section 4 we will show how the framework supports each pattern of flexibility.

## 4 Achieving Flexibility

In this section we demonstrate how the framework elements can be used to design a flexible process. For purposes of illustration we refer to a social service process from the charity domain that has been modelled in our object-oriented approach, which is presented in Figure 4. This model consists of a Client Intake COROB that manages the process of accepting new clients who have contacted the charity for assistance. This COROB is responsible for creating and coordinating the tasks and sub-processes involved in new client intake such as completing a risk assessment, visiting the client and collecting reports from social workers, whilst also coordinating distribution of major issues to other COROBs. The model captures several points in the process where flexibility is either allowed or constrained. For example, a referral to a Homelessness COROB can be performed at any time in the Review Region but at no other time. To counter the possibility of a variety of exceptional circumstances arising at runtime the model has been designed to capture the creation, delegation and nesting patterns of flexibility.

The rest of the section uses extracts of the process model shown in Figure 4 in order to discuss how the framework addresses the three patterns of flexibility.

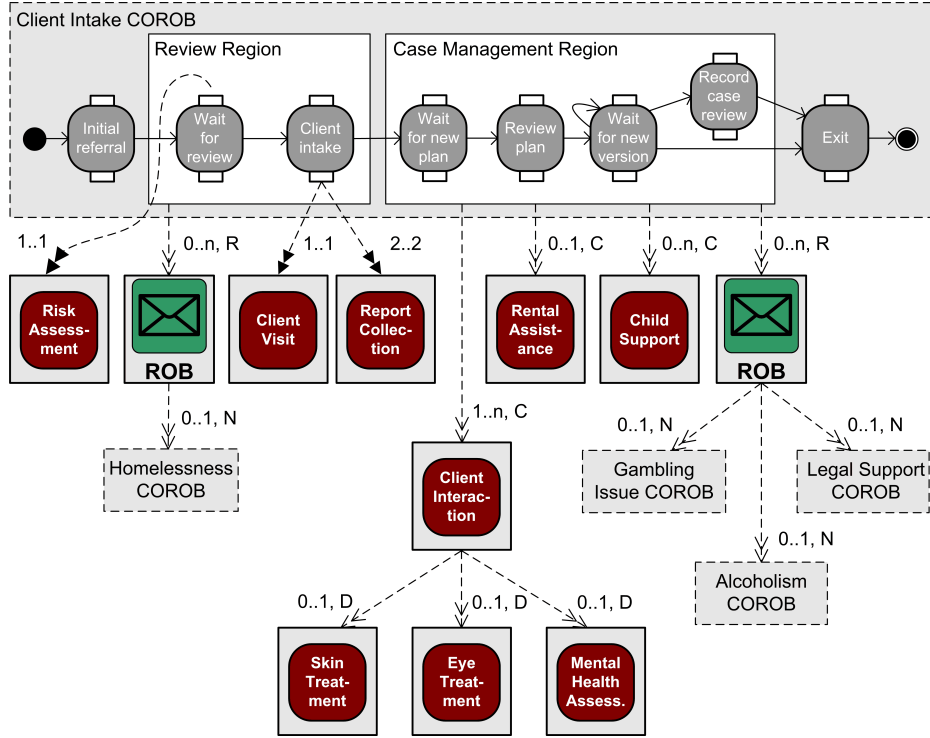


Fig. 4. Object-Oriented Social Services Delivery Model

#### 4.1 PoF1: Creation Flexibility

Creation flexibility is achieved by specifying the set of TOBs can be created on-demand by defining a creation region within a COROB then linking the creation region to those TOBs with the *creation signal*, as shown in Figure 5. In this example a social worker tailors a plan for a client to resolve the issue(s) that the client is faced with. Since the plan is tailored to the unique circumstances of an individual, the plan for each client is almost always different. To operationalise the plan the social worker then requires access to different tasks offered by the charity (represented by the TOBs). Creation flexibility gives the social worker the ability to create instances of a task when it is needed (i.e. in any of these states: “Wait for new plan”, “Review plan”, “Wait for new version” and “Record case review”), rather than when it is planned.

When the Client Intake COROB is in a state contained in the Case Management Region,  $1..n$  instances of the Client Interaction TOB,  $0..n$  instances of

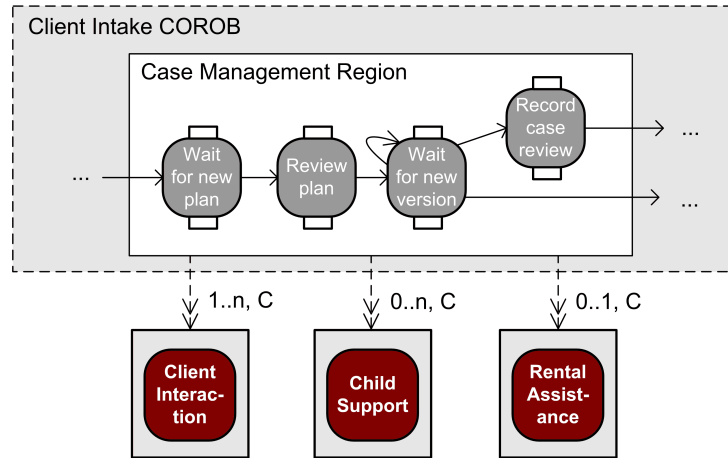


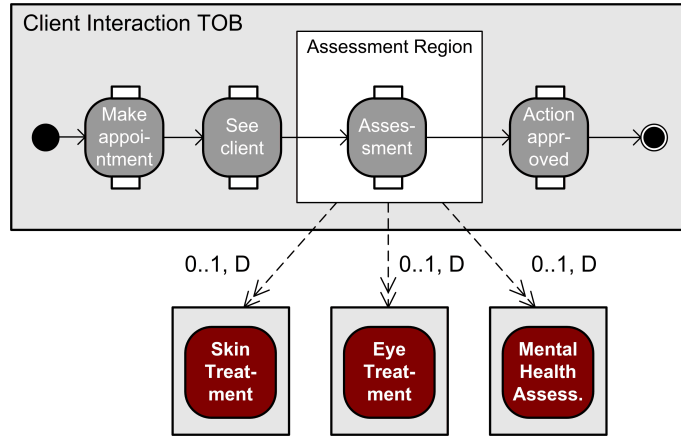
Fig. 5. Creation Pattern of Flexibility

the Child Support TOB and  $0..1$  instances of the Rental Assistance TOB can be created. At least one Client Interaction TOB *will* be created before exiting the Case Management Region, but more than one instance *may* be created. Any number of Child Support TOBs along with a maximum of one Rental Assistance TOB *may* be created. Creation flexibility allows a designer to capture on-demand task creation while also constraining the type and number of task instances according to the business rules.

#### 4.2 PoF2: Delegation Flexibility

Delegation flexibility is achieved by linking a creation region in a TOB to one or more tasks using the *delegation signal*. In Figure 6, we demonstrate delegation using the Client Interaction delegator TOB. This TOB contains three states (“Make appointment”, “See client” and “Assessment”) and one creation region (named “Assessment Region”) that contains the “Assessment” state. This creation region imposes two restrictions on the Client Interaction TOB. Firstly, delegation from a Client Interaction can only be performed when it is in the Assessment Region. Secondly, the set of allowable delegatee tasks from this creation region are the *Skin Treatment*, *Eye Treatment* and *Mental Health Assessment* TOBs which are subtypes of the Client Interaction TOB.

Delegation is an optional action – a user will make the choice at runtime of whether or not delegation is performed because the multiplicity of each delegation signal is  $0..1$ . If a delegator has more than one delegatee then a choice is made by the user to select which TOB will become the delegatee. Delegation can never be mandatory, i.e. a delegation signal must have a lower bound of 0. Delegation is not allowed if the upper bound is greater than 1 because this implies creating clones of the delegator. If multiple instances of a delegator are needed they would firstly be created and then permitted to delegate as required.



**Fig. 6.** Delegation Pattern of Flexibility

In case delegation does not occur during the execution of a delegator then its execution will complete normally.

This example illustrates how object inheritance is used to capture delegation associations between tasks in a process model. However we point out that delegation extends the concept of inheritance since at runtime a delegatee must take the data and context of the delegator and must also complete its lifecycle in the same way that the delegator would have.

### 4.3 PoF3: Nesting Flexibility

Nesting flexibility is achieved by linking a creation region in a COROB to a ROB using the *referral signal*, then linking a creation region in the ROB to one or more COROBs using the *nesting signal*. At runtime, a parent COROB may invoke the referral signal to create an instance of a ROB. The ROB may invoke a nesting signal to create an instance of a child COROB to manage the newly discovered real-world issue. The type of child COROB to create is determined by a user. The ROB creates two levels of indirection between the parent and child COROB, giving the framework two advantages.

Firstly, COROBs are decoupled, which establishes COROB modularity. Secondly, the ROB provides the opportunity for human intervention in a referral, since referring major issues between in this manner often needs an approval from a third party (e.g. a manager), who can either permit or deny creation of a new COROB instance. Hence, the ROB behaves as an arbiter that separates a parent COROB from its children, allowing children to execute in parallel and allowing a manager to maintain control over nested COROBs.

In Figure 7 we see the number of referral signals that may be sent from the Case Management Region to a ROB is unbounded ( $0..n$ ) and the ROB is connected to three COROB types. For example, if a social worker discovers an

alcoholism issue with a client, a ROB will be created in the system which will in turn create an Alcoholism COROB instance. Alternatively, if an alcoholism and gambling issue are discovered with a client the system will create two ROBs and (given management approval) one ROB will create an Alcoholism COROB instance and the other ROB will create a Gambling Issue COROB instance.

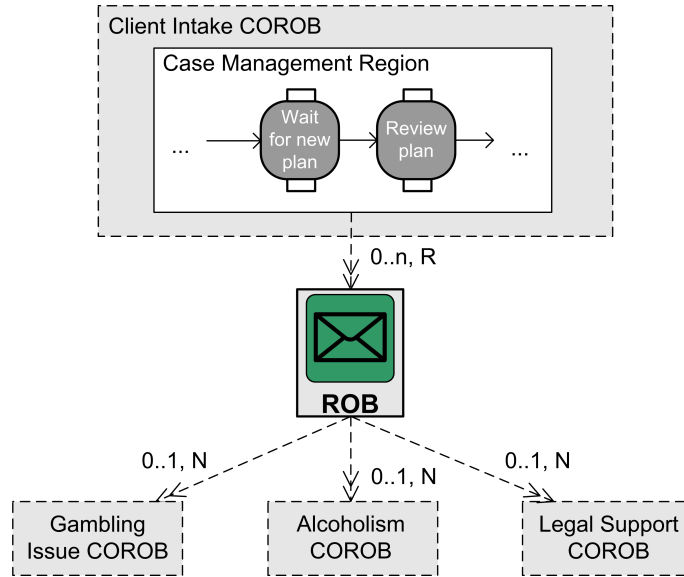


Fig. 7. Nesting Pattern of Flexibility

The framework places no restrictions on the levels of nesting meaning that a child COROB can in turn create its own ROBs, which can create their own COROBs and so on. For example, as shown in Figure 8, in the “Wait for new plan” state an issue resolution plan could be prepared for an unemployed client which identifies a new issue beyond the scope of the Client Intake COROB. The issue is referred to a nested Work Search COROB. However, during the execution of the Work Search COROB the client unexpectedly falls into serious trouble with the police. The Work Search COROB creates a new ROB, which creates a nested Legal Support COROB to support the clients unemployment issue.

We observe that the main benefit of nesting flexibility for a user is the ability to call in different sets of resources and skills in response to situations as they arise. Nesting flexibility allows a COROB to maintain control over the type and number of all dependent COROBs without being directly linked to them, while also establishing an unplanned structure of nested processes.

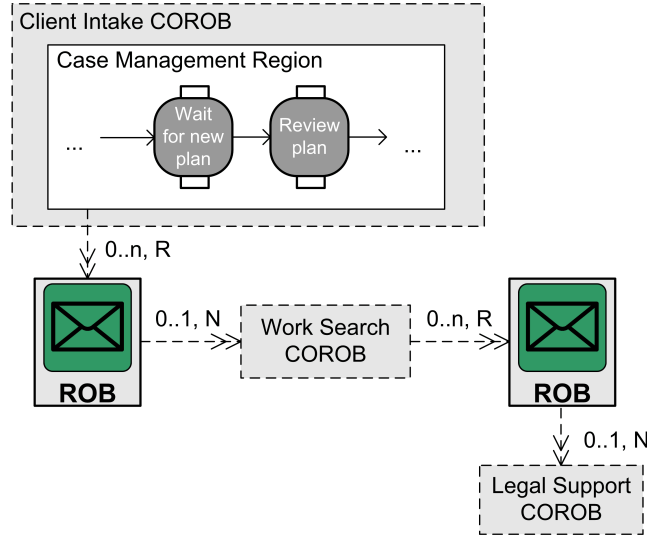


Fig. 8. Nested Unplanned Sub-processes

#### 4.4 Putting it all together

Using the examples in this section we have demonstrated how an OO process model can handle issues in an unplanned manner. The modelling notation is based on an OO meta-model that has been designed to approach exceptional circumstances as they occur by engaging creation, delegation and nesting flexibility. The ability to handle work in the different ways that it may appear is the point of distinction that allows several flexibility requirements that were identified in Section 2 to be supported. The concept of creation regions in particular enables a designer to clearly define which types of flexibility are related to which set(s) of states. This approach gives a process model designer the ability to express that flexibility *is* required at particular points and that flexibility *is not* required at other points, which is beneficial for the design of flexible process models.

A modelling tool named *FlexConnect*<sup>5</sup> has been developed that allows us to design OO process models as described in this paper. A formal semantics of the proposed meta-model has also been defined in terms of a Coloured Petri Net (CPN) [7]. The modelling tool has an export capability to provide a CPN Tools input file that contains an initial marking for the CPN. In addition to providing a formal grounding to the proposal, the mapping to CPN provides tool support for analysis of FlexConnect process models. The modelling tool, export function and the generated CPNs have been tested with 20 sample OO process models of varying sizes including the example shown in Figure 4.

<sup>5</sup> FlexConnect can be downloaded from <http://code.google.com/p/flexconnect/>

## 5 Related Work

There has been a significant amount of research related to *flexible process management*. Research in this field has focused on dealing with runtime deviations with respect to the expected execution of a process model (*dynamic change*). A proposal consisting of five criteria for characterizing dynamic change [8] shed some light into the shortcomings of conventional process management systems with respect to flexible execution, and enabled comparative evaluation of the change-handling capabilities of a number of process management systems. Weber et al. [3] built on top of this work by defining 17 change patterns. The authors advocate that (ideally) there should be alignment between computerised and real-world processes, a position shared by work done on ADEPT<sub>flex</sub> [9] and also our proposal that has demonstrated how work can be handled as it occurs in an *unplanned* manner.

DECLARE [2] is an example of a Constraint-Based Workflow Modelling tool that describes loosely-structured processes using a declarative approach that allows a process designer to focus on the ‘what’ rather than the ‘how’. The strength of this approach is that model constraints can be added or relaxed where needed. Our framework includes the definition of *creation regions* in which object types (or subtypes thereof) can be created within cardinality restrictions - a capability that goes beyond what the DECLARE framework offers.

A taxonomy of process flexibility identified and defined four types of flexibility [10]. These types are flexibility by design, flexibility by change, flexibility by deviation and flexibility by underspecification. Using this taxonomy it may be observed that our framework supports a spectrum of flexibility types, for example delegation flexibility is flexibility by design, creation flexibility is flexibility by deviation and nesting flexibility is flexibility by underspecification.

The “Flexibility as a Service” (FAAS) proposal [11] is a structured approach inspired by the taxonomy of flexibility that enables a process designer to combine the flexibility aspects of three process modelling approaches, namely YAWL [12], DECLARE [2] and WORKLETS [13]. We have shown in this proposal how to design flexible process models using existing OO modelling techniques as an alternative to mixing-and-matching process modelling languages.

Work by Klingemann observed that true benefits are gained from flexibility in process models when flexibility is controlled [14], an observation we agree with. To achieve flexibility in process models Klingemann classified three flexible element types; alternative activities, non-vital activities and optional execution order. Our framework extends this classification to cater for additional types of process intervention such as task delegation and creation regions.

Other OO or object-based process modelling approaches have been proposed by Küster et al. [15] and Wirtz et al. [16]. However, these latter proposals are not motivated specifically by flexibility requirements. For instance, the work of Küster et al. is instead motivated by compliance management. An alternative paradigm to OO process modelling is *case handling* [5], a paradigm for business process support where the primary focus is on the data supporting a system rather than purely on capturing control-flow behaviour. It was suggested that

shifting focus away from pure control-flow can lead to less restrictive systems. This view is also supported by Weske et al. [1] and Müller et al. [17] who have proposed alternative object-oriented process modelling approaches.

Some parallels can be drawn between the concept of a Coordination Object (COROB), and the “multiple instance without *a priori* runtime knowledge” workflow pattern [18]. Parallels may also be observed between the concept of a Referral Object and proposals such as WORKLETS, which provides users with a method to dynamically respond to change by taking action not originally envisaged as part of the control-flow behaviour. The proposal in this paper puts these concepts together and grounds them in an OO process modelling approach.

## 6 Conclusions and Future Work

In this paper we demonstrated how a small set of coordination concepts, in combination with established object-oriented modelling techniques, enables the design of highly flexible processes consisting largely of *unplanned* activities. We demonstrated in particular how a small set of object types (i.e. Coordination Object, Task Object and Referral Object) can be combined to capture different patterns of flexibility. The key principle is that a Coordination Object defines “what can happen during a case”, rather than “how should it happen”. Any constraints regarding which objects can or should be created and when, are overlaid on top of the basic object model. This is in contrast with mainstream process modelling paradigms based on flowchart-like notations, in which the activities to be performed and their control-flow relations form the backbone of a process model.

In particular we have focused on capturing flexibility aspects as it pertains to creating new objects to perform unplanned activities at design-time. Of course, while flexibility is essential in domains such as human services, there are situations where this flexibility should be constrained. The proposed framework supports the definition of simple ‘thresholds’ to constrain the minimal and maximal number of TOB and ROB objects of various types that should be started under a COROB of a given type (cf. the multiplicity constraints of a signal).

In addition to this feature, one may need to define more sophisticated constraints. For example, we have encountered situations that require the definition of “creation regions”, to establish when instances of a given TOB or ROB type can be created under a COROB of a given type – e.g. a ROB corresponding to “Work Search” COROB should only be started after the “Health Treatment” tasks have completed. Also, we have encountered situations where one needs to constrain the number of TOBs or ROBs of different types that need to complete before a COROB object moves to a completion state – e.g., a COROB to handle a case for a homeless family will not complete until the process created to deal with their homelessness situation has closed. The definition of such synchronization constraints within the proposed framework is ongoing work.

**Acknowledgement** This work is supported by an Australian Research Council Linkage grant (LP0562363) co-funded by FlowConnect Pty Ltd.

## References

1. Weske, M.: Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In: 34th Annual Hawaii International Conference on System Sciences (HICSS-34), Maui, Hawaii (2001)
2. Pesic, M., Schonenberg, M., Sidorova, N., van der Aalst, W.: Constraint-Based Workflow Models: Change Made Easy. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA and IS. (2007) 77–94
3. Weber, B., Rinderle, S., Reichert, M.: Change Patterns and Change Support Features in Process-Aware Information Systems. In: 19th International Conference on Advanced Information Systems Engineering, Trondheim, Norway (2007) 574–588
4. Dadam, P., Reichert, M., Rinderle, S., Jurisch, M., Acker, H., Göser, K., Kreher, U., Lauer, M.: Towards Truly Flexible and Adaptive Process-Aware Information Systems. In: Information Systems and e-Business Technologies, 2nd International United Information Systems Conference, Klagenfurt, Austria (2008) 72–83
5. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. *Data and Knowledge Engineering* **53** (2005) 129–162
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, Boston, MA, USA (1995)
7. Jensen, K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Springer-Verlag (1997)
8. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems - a survey. *Data and Knowledge Engineering* **50** (2004) 9–34
9. Reichert, M., Dadam, P.: ADEPT<sub>flex</sub>-Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems (JIIS)* **10** (1998) 93–129
10. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.: Towards a Taxonomy of Process Flexibility. In: Proceedings of the CAiSE'08 Forum, Montpellier, France (2008) 81–84
11. van der Aalst, W., Adams, M., ter Hofstede, A., Pesic, M., Schonenberg, H.: Flexibility as a Service. Technical Report BPM-08-09, BPMcenter.org (2008)
12. van der Aalst, W., ter Hofstede, A.: YAWL: Yet Another Workflow Language. *Information Systems* **30** (2005) 245–275
13. Adams, M., ter Hofstede, A., Edmond, D., van der Aalst, W.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA and ODBASE, Montpellier, France (2006) 291–308
14. Klingemann, J.: Controlled Flexibility in Workflow Management. In: Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden (2000) 126–141
15. Küster, J., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: Proceedings of the 5th International Conference on Business Process Management (BPM), Brisbane, Australia (2007) 165–181
16. Wirtz, G., Weske, M., Giese, H.: The OCoN Approach to Workflow Modeling in Object-Oriented Systems. *Information Systems Frontiers* **3** (2001) 357–376
17. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA and IS. (2007) 131–149
18. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. *Distributed and Parallel Databases* **14** (2003) 5–51