A Flexible, Object-centric Approach for Business Process Modelling

Guy Redding

Queensland University of Technology, Brisbane, Australia

g.redding@qut.edu.au

Marlon Dumas

University of Tartu, Tartu, Estonia

marlon.dumas@ut.ee

Arthur H. M. ter Hofstede

Queensland University of Technology, Brisbane, Australia

a.terhofstede@qut.edu.au

Adrian Iordachescu

FlowConnect Pty Ltd, Sydney, Australia

adrian@sws.com.au

Mainstream business process modelling techniques often 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. The framework is presented in this article and we show how it addresses different flexibility requirements using a series of illustrations.

object-centric; object-oriented; process modelling; flexible workflow.

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 (e.g. [1-4]). In this article we demonstrate how to capture highly flexible business processes using an object-centric (O-C) process modelling approach. The approach is inspired by, but arguably not limited to, the delivery of human and social services.[1] Modelling and executing processes in this domain presents additional challenges 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, variations on a case-by-case basis and exceptions are the norm in these processes. 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.

In this article, we explore the hypothesis that an object-centric modelling approach 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 O-C process models.

2 Patterns of Flexibility

In our experience in applying object-oriented approaches to design process-aware systems that need to deal with ad-hoc situations, a range of requirements have been observed that are 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 these patterns of flexibility are referred to as PoF1-PoF3.

[1] This work is inspired by a project involving the fourth author.

2.1 PoF1: Creation Flexibility

Creation flexibility is the ability of a user to trigger the creation of one or more task instances (*jobs*) in an unplanned manner during execution of a process. This pattern of flexibility allows the set of task types to be instantiated as well as the ordering of instantiations to be loosely specified at design-time. Creation flexibility is similar to the case handling approach [18] where tasks do not need to be performed in a strict order and do not necessarily have to be completed to complete a case (meaning that the tasks are optional). 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 *as-specified* by process model logic. An *unplanned* task presents additional concerns since it 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 subtypes of *Treatment*, but the additional treatments are difficult to completely plan at design-time because the treatment(s) depend on the assessment.

2.2 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 provides support for circumstances that may change over time (i.e.\ if a problem appears during a client interaction, delegate the interaction to a task that can support the problem). 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.

To support such situations, a new task (*delegatee*) takes over execution of a previous task (*delegator*). For the purposes of control-flow, a delegatee replaces a delegator, meaning that when a 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.This feature, along with the fact that data is transferred from a delegator to a delegatee, distinguishes delegation flexibility from creation flexibility. 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.

2.3 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 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 -- we call this situation 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, while maintaining sub-process modularity and retaining process control.

3 Elements for Flexible Object-centric Processes

We aim to fulfill the objective of achieving flexibility in object-centric models by proposing a design framework consisting of three abstract types of business objects, namely the Coordination Object, Job 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.
We propose to achieve process flexibility via an extended meta-model that consists of three abstract types of business objects, namely Coordination Object, Job Object and Referral Object. As shown in Figure 1, a concrete business object type inherits from an abstract type.

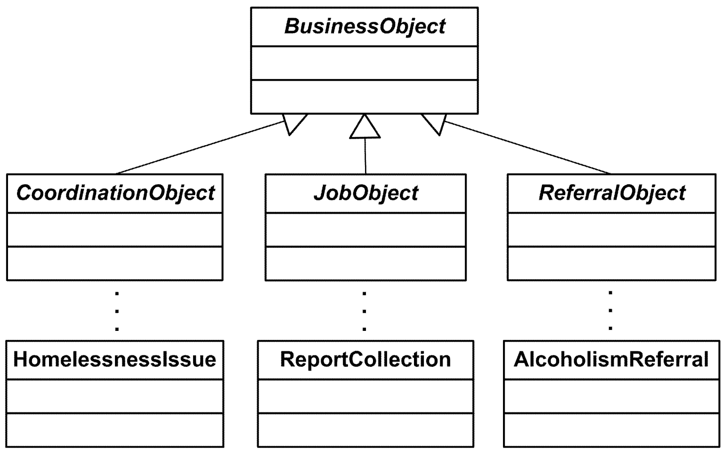

Figure 1. Abstract Types and Concrete Types

A **Coordination Object** (COROB) is an object that *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 execution of a process as well as referring out of scope work to other COROBs.

A **Job Object** (JOB) is an object that represents a task. A JOB manages task execution and reports task completion to its parent object. For example, two JOBs 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 a COROB.

The base meta-model of object types and their relations can be found in our previous work [22], which has been captured using the ORM notation [24]. The flexibility extensions to the base meta-model are also captured using ORM and are presented in Figure 2. We now introduce the base meta-model extensions captured by this ORM.

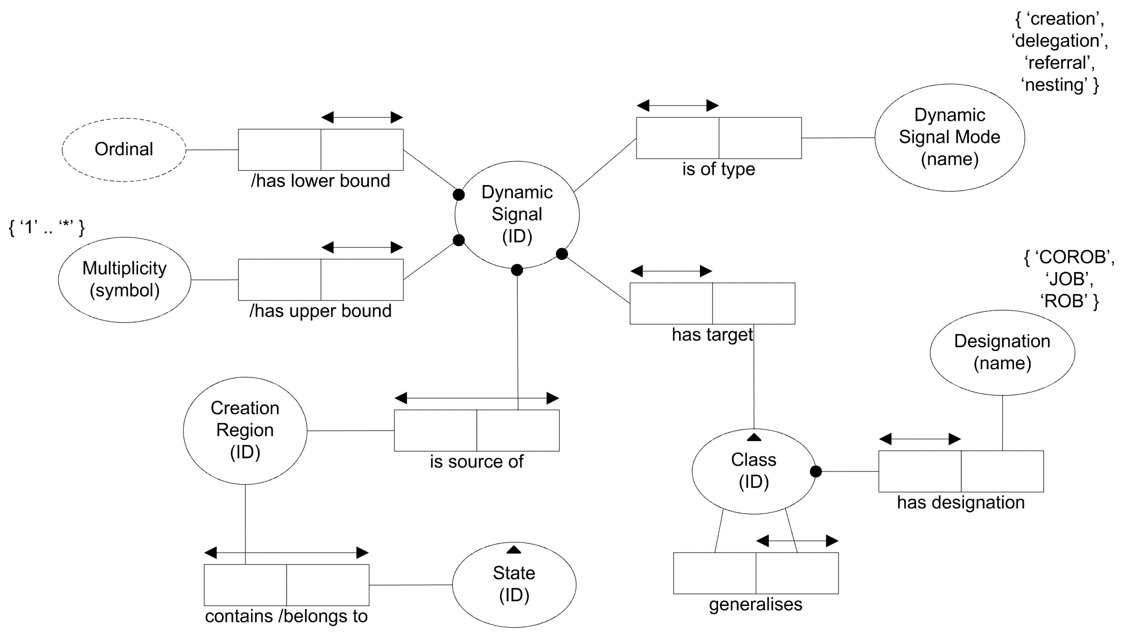

Figure 2. ORM for Flexibility Extensions

An *object-centric process model* consists of a set of object types (COROB, JOB and ROB subtypes) and their relations. Every object type specified in a model is a subtype of one of the three base object types: COROB, JOB or ROB. For example, a "Homelessness Coordination Object" is a COROB subtype and a "Client Appointment" is a JOB subtype. A subtype relation is established by using a generalisation association. Generalisation is a classical object-oriented concept that allows a subtype to inherit attributes and behaviour from a supertype. In the case of object-centric process models we may make use of generalisation to organise common process-related attributes and behaviour in a hierarchy of objects. For example, a "Skin Treatment", "Eye Treatment" and "Mental Health Assessment" JOBs are subtypes of a "Treatment" JOB. The generalisation association allows a supertype to delegate its lifecycle to a subtype at runtime and requires that each subtype sends and receives the same signals as a supertype and sends and receives at least the same data as its supertype, while allowing the subtype to capture an object lifecycle that specialises the supertype. Since the correct application of behavioural specialisation of object lifecycles (i.e. ensuring that inheritance does not lead to behavioural inconsistencies) is a separate research question and has been covered in works (for example) by Schrefl and Stumpter found in [25] and [26], we do not elaborate any further on this topic.

A *creation region* is a collection of one or more states in a state machine from within which it is possible to create object instances from a set of object types. A state can belong to more than one creation region, but those states must belong to the same state machine. From a creation region, any number of dynamic signals can be sent. 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 choose to 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 with the dynamic signal and input given by the user when triggering the dynamic signal. This approach follows the principle of the Strategy Pattern [7].

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 JOB to a target delegatee JOB. 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 JOB 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 JOB. 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. In Figure 3, we show how the COROB, JOB and ROB can be connected using the four dynamic signal types to capture the three PoFs.

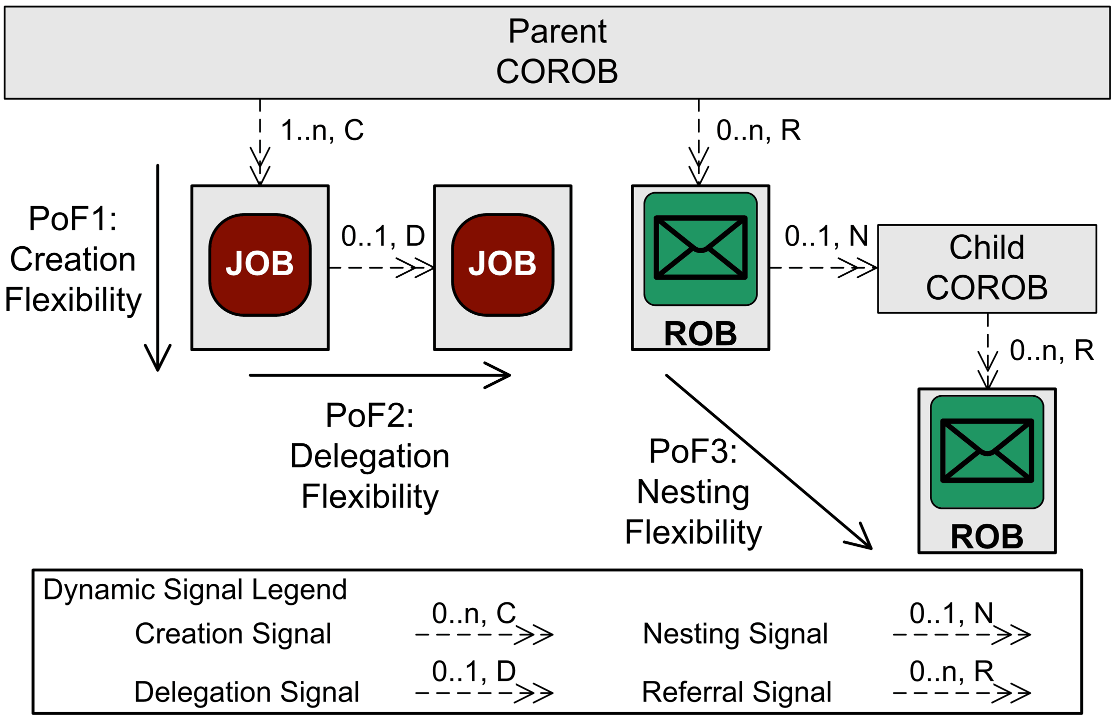

Figure 3. Patterns of Flexibility in the Framework

4 Working Example -- Social Service Provision

As a motivating scenario, we consider a process executed in the context of a charity organisation. 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 the 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. The elements of the framework are represented graphically using the notation in Figure 4.

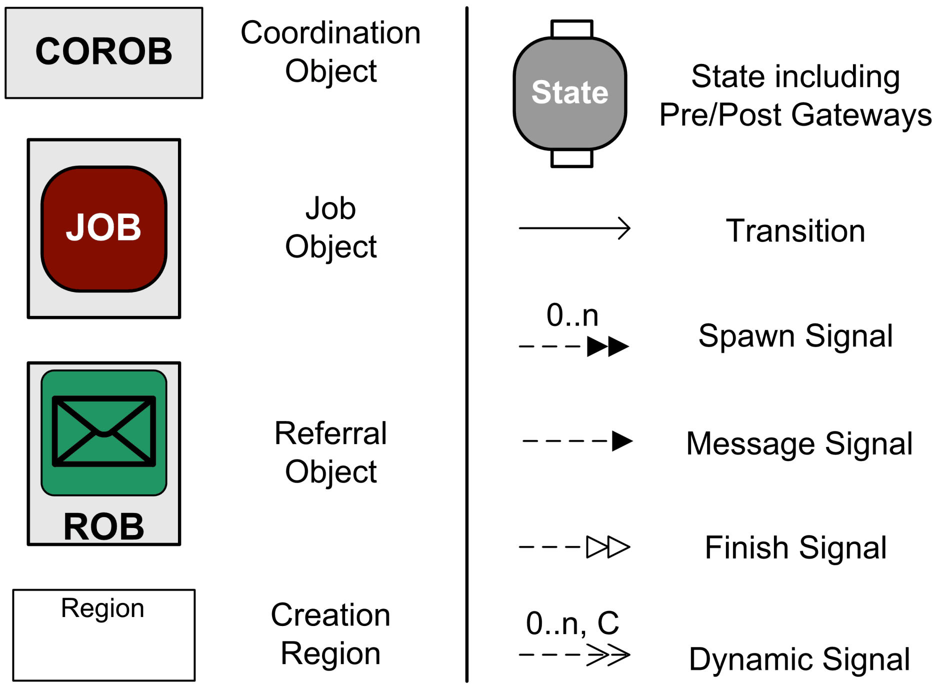

Figure 4. Extended Object Model Elements

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 for a charity organisation that has been modelled using the object-centric approach presented in this article, which is presented in Figure 5. This model consists of a Client Intake COROB that manages the process of accepting new clients who have contacted the charity for assistance. The 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 5 in order to discuss how the framework addresses the three patterns of flexibility.

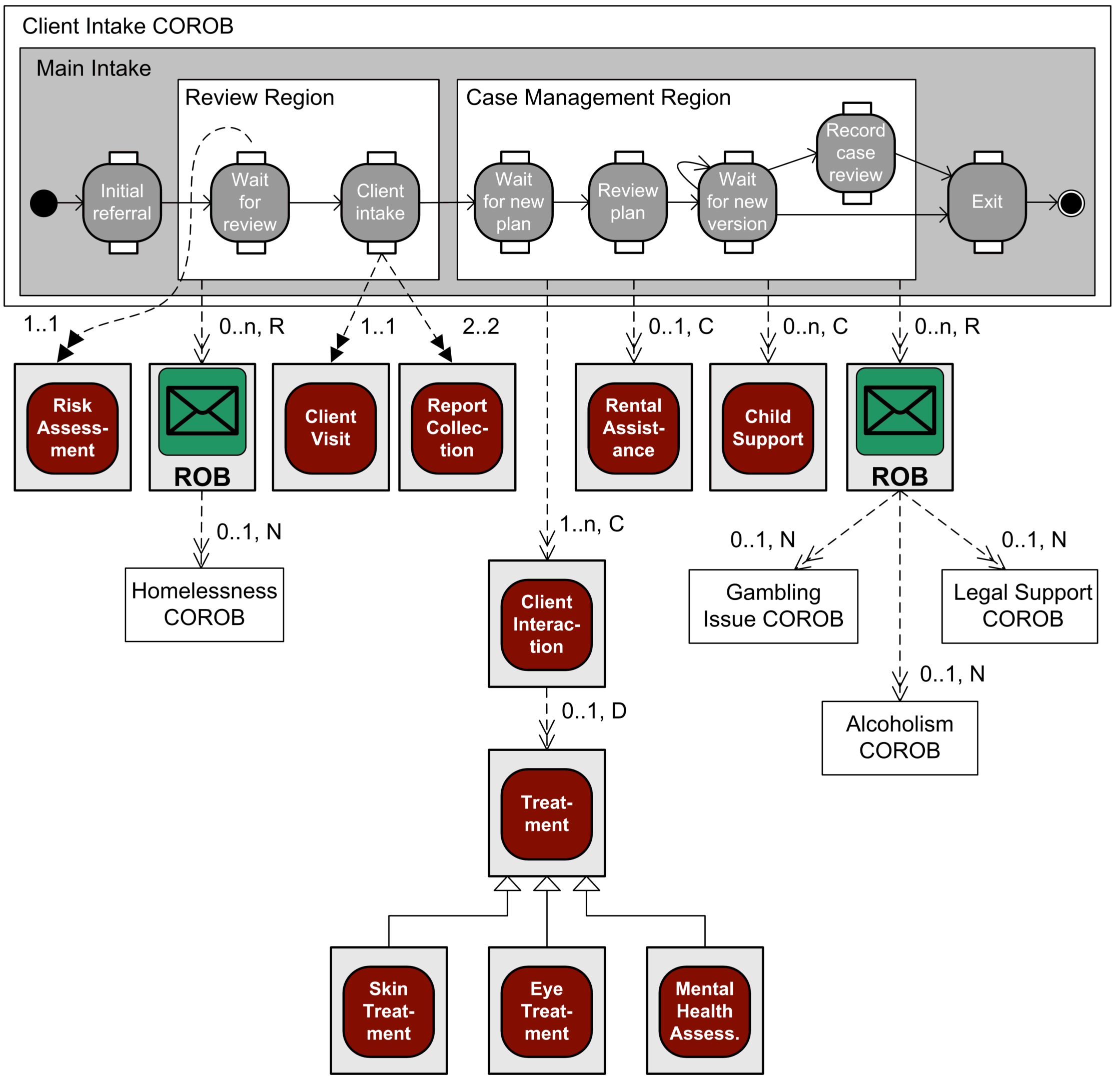

Figure 5. Object-centric Social Services Delivery Model

4.1 Demonstrating Creation Flexibility

Creation flexibility is achieved by specifying the set of JOBs that can be created on-demand by defining a creation region within a COROB then linking the creation region to those JOBs with the *creation signal*, as shown in Figure 6. 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 JOBs). 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.

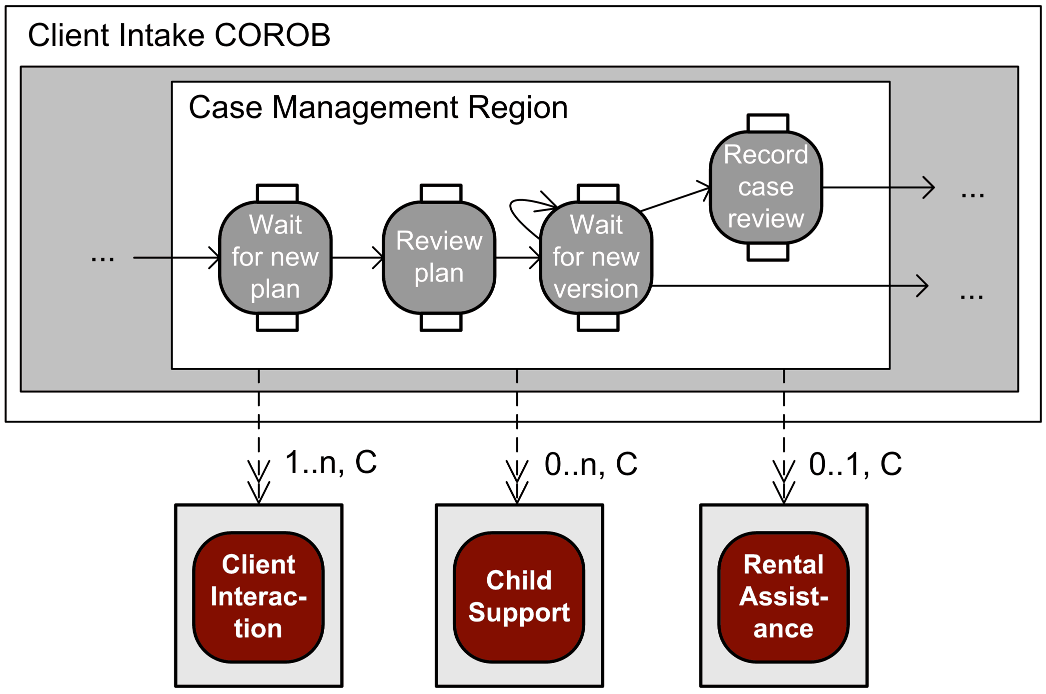

Figure 6. Creation Pattern of Flexibility

When the Client Intake COROB is in a state contained in the Case Management Region, *1..n* instances of the Client Interaction JOB, *0..n* instances of the Child Support JOB and *0..1* instances of the Rental Assistance JOB can be created. At least one Client Interaction JOB *will* be created before exiting the Case Management Region, but more than one instance *may* be created. Any number of Child Support JOBs along with a maximum of one Rental Assistance JOB *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 Demonstrating Delegation Flexibility

Delegation flexibility is achieved by linking a creation region in a JOB to one or more tasks using the *delegation signal*. In Figure 7, we demonstrate delegation using the Client Interaction delegator JOB. This JOB 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 JOB. 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* JOBs which are subtypes of the Treatment JOB.

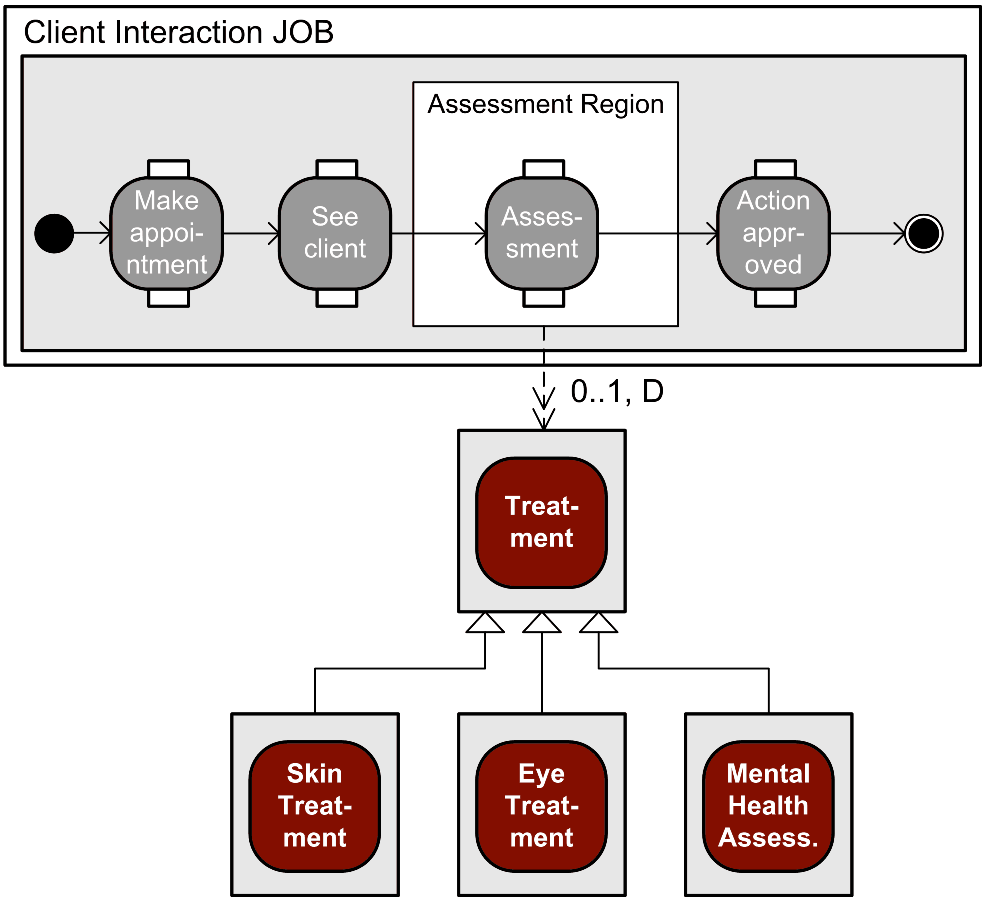

Figure 7. Delegation Pattern of Flexibility

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 JOB 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. 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 Demonstrating 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 resource (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 third party resource to maintain control over nested COROBs.

In Figure 8, we see that the number of referral signals that may be sent from the Case Management Region to a ROB is unbounded (*0..n*) and that 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 and the other will create a Gambling Issue COROB.

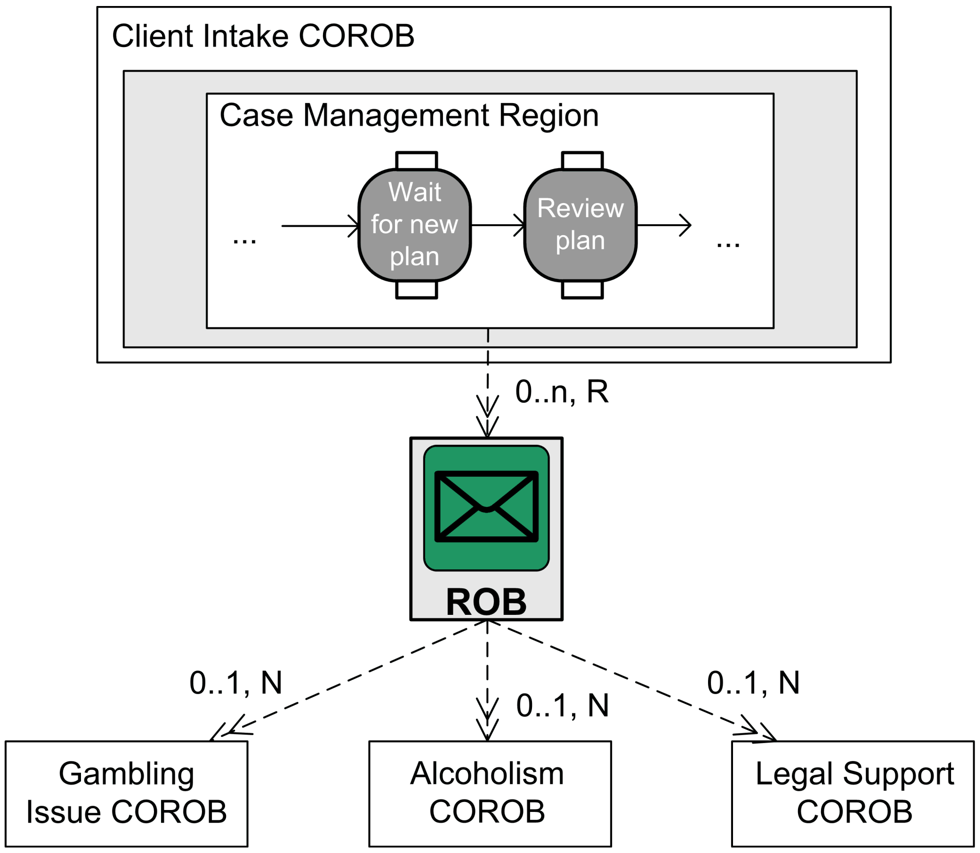

Figure 8. Nesting Pattern of Flexibility

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 9, in the "Wait for new plan" state an issue resolution plan is prepared for an unemployed client which identifies an unemployment issue beyond the scope of the Client Intake COROB. The issue is referred to a nested Work Search COROB. However, during 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.

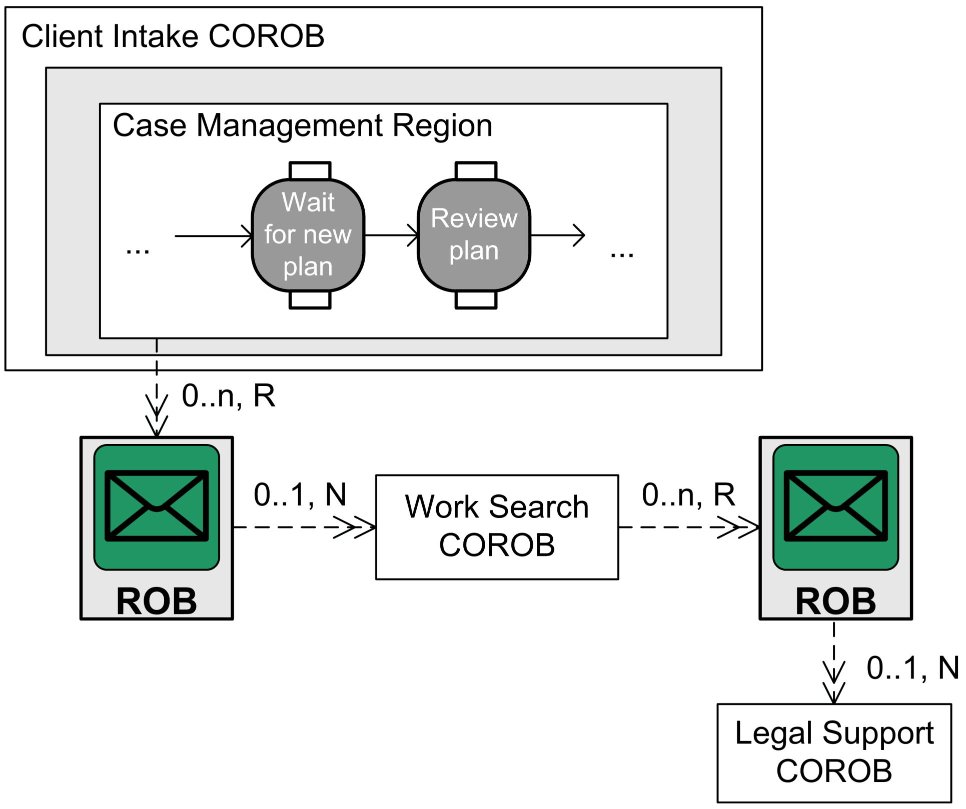

Figure 9. Nested Unplanned Sub-processes

Using the examples in this section we have demonstrated how an O-C process model can handle unplanned tasks and issues. The modelling notation is based on an object behaviour 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 which allows several flexibility requirements to be supported, as identified in Section 2. 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. In the next section we present a tool called *FlexConnect* that supports modelling of flexible object-centric models as presented in this article.

5 Tool Support

A modelling tool named *FlexConnect* has been developed that allows us to design O-C process models as described in this article.[2] FlexConnect is a tool that assists process designers to develop O-C process models and was developed using the Eclipse Graphical Modelling Framework (GMF). The foundation of the tool is the UML Class diagram shown in Figure 10 that captures the FlexConnect GMF Domain Model. The GMF Domain Model is a specification of the modelling tool elements and their associations.

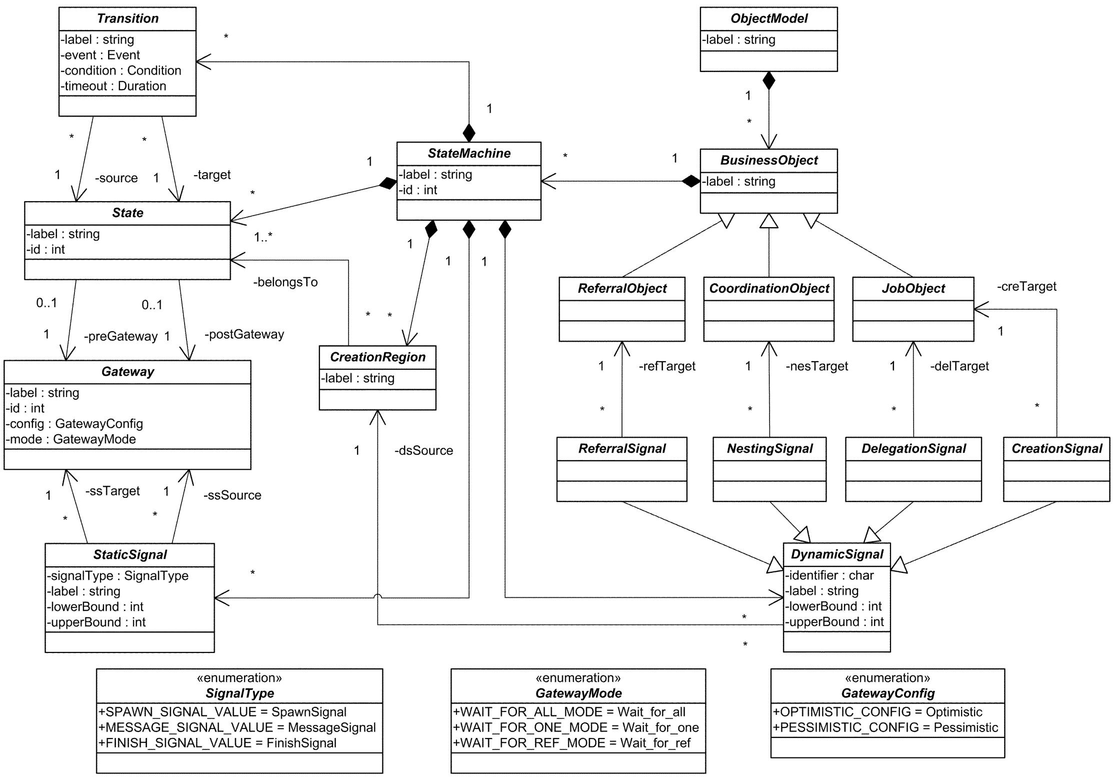

Figure 10. UML Class Diagram for Object-centric Flexibility

The modelling tool has a feature that generates and exports an initial marking to a file that is used as input to a Coloured Petri Net (CPN) [8], which is available with the FlexConnect tool. The CPN was developed using the CPN Tools software to provide us with the ability to formally check, validate and simulate the behaviour of models that have been designed using FlexConnect. The modelling tool, export feature and the generated CPNs have been tested with 20 sample O-C process models of varying

[2] FlexConnect can be downloaded from http://code.google.com/p/flexconnect/

sizes in order to evaluate the behaviour of the elements of the base model as well as validate each pattern of flexibility. This includes the social services example presented in this article (see Figure 5), which is shown as a FlexConnect model in Figure 11.

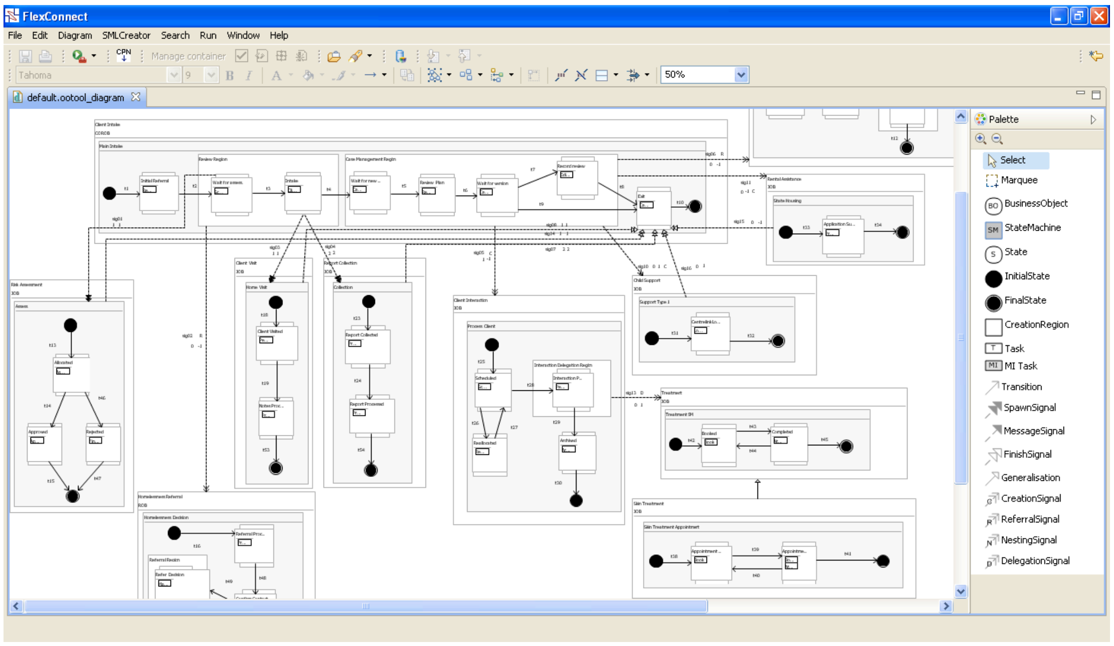

Figure 11. Social Services Model in FlexConnect

We will now walk through this social services support model shown in Figure 5 at runtime. Upon entering the Review Region the “Wait for review” state is entered. A Risk Assessment JOB is completed for the applicant while an initial application is being completed. At this stage it is either confirmed or not that the client has a Homelessness Issue. A Homelessness Issue is a major issue that requires management by a separate COROB that was designed to manage such an issue. If a Homelessness Issue is confirmed, the Main COROB refers this new work out to a ROB which creates a nested instance of a Homelessness COROB.
Following creation, the Homelessness COROB will execute in parallel to the Main COROB, creating its own tasks that manage the needs of the client to do with their homelessness issue. During the execution of the Homelessness COROB an additional issue is discovered with the client to do with a drug dependency. The Homelessness COROB reacts to this issue by invoking a referral. The ROB is guided by the user to create a nested instance of a Drug Dependency COROB that executes in parallel to

the Homelessness COROB. This parallelism is handled in a structured manner due to the concept of nesting flexibility.

After the “Client intake” state is entered, three tasks are created. A Client Visit JOB is created along with two Report Collection JOBs. The Client Visit manages the procedure of a social worker's visitation to a client, while the Report Collection manages the work involved with reporting on the recovery progress of a client.

After exiting the “Client intake” state the Review Region is exited and the Case Management Region is entered. This region consists of four states, which are: “Wait for new plan”, “Review plan”, “Wait for new version” and “Record case review”. In any state of the Case Management Region we have the ability to create *1..n*, on-demand, Client Interaction tasks. Specifically, at least one Client Interaction TO *will* be created before the Case Management Region is exited, but more *may* be created. This is an example of creation flexibility. In the “Wait for new plan” state a social worker prepares a goal-action plan for the client, which is revised in the “Review plan” state, and a Client Interaction TO is created by the social worker to suit the social workers need to approach the client with clarifications regarding the case. During the interaction with the client the social worker finds that the client needs additional medical care and the Client Interaction is delegated to a Skin Treatment. Here we see an example of delegation flexibility.

During the “Wait for new version” state a major alcoholism-related issue is discovered. To handle this situation an instance of an Alcoholism Issue COROB is created. The creation of this new COROB is performed using the same method as the Homelessness Issue COROB, as this method allows us to manage the uncertainty surrounding the unknown and unpredictable runtime aspects of the process. These unknown aspects are the elements of a process that *may* be invoked, such as an alcoholism issue in this case. The motivation behind supporting the invocation of process elements in this manner is due to the unknown aspects of *if* and *when* during the execution of a Homelessness Issue (and indeed, any other process that supports a social service) that may be encountered.

During the “Record case review” state another major issue is discovered with the client and an instance of a Gambling Issue COROB is created to handle the issue. The ability to handle work in the different ways that it may appear is the point of distinction that allows the flexibility requirements that were identified in Section 2 to be supported.

The output of a valid model constructed using the FlexConnect modelling tool is a Standard ML (SML) [27] file. An SML file generation feature is found on the FlexConnect toolbar that creates an SML file from an O-C model by pressing a button named "SML Creator". Upon pressing this button, the syntax of the object model is validated. To avoid creating an invalid SML file the O-C model must pass a series of validation checks. If one or more of the checks are not passed, a list of the problems that were found in the model are presented in a popup box and an SML file is not created. Otherwise, the result is reported in a popup box and an SML file is created. The checks that are performed on a model include:

- The names of all nodes except Tasks (State Machines, States, Gateways and Creation Regions) must be unique and non-null.
- The names of all connections (Transitions, Static Signals and Dynamic Signals) must be unique and non-null.
- The upper bound of all (static and dynamic) signals must be greater than or equal to the lower bound.
- The upper bound of all multiple instance tasks must be greater than or equal to the lower bound.
- Each gateway must have a configuration and a mode.
- Each message signal and finish signal must have either a parent spawn signal or parent dynamic signal.

An SML file created by the FlexConnect modelling tool contains an initial marking for the following places in the CPN: Signal Connections, Gateway Mode, Gateway Configuration, State Gateways, Transitions, Creation Regions, Dynamic Connections, Generalisation Associations and Tasks. Each place is populated by making a call to a function in the SML file. E.g. the Transitions place calls the *getTransitions()* function, which places a single token in the Transitions place that contains a list of the transitions in the O-C process model. Successfully loading the SML file into the CPN without receiving any error reports indicates that the O-C process model is at least syntactically correct, because the type of each place in the CPN is directly mapped to a concept in the O-C meta-model. For example, the "Dynamic Signal Connections" place contains a list of the dynamic signals in the O-C process model and the "Creation Regions" place contains a list of the creation regions.

6 Related work

There is 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 framework comprising five criteria for characterizing dynamic change [9] shed some light into shortcomings of conventional process management systems, and enabled comparative evaluation of the change-handling capabilities of process management systems.
Weber et al. [3] built on top of this work by defining 17 change patterns. The authors advocate that there should be alignment between computerised and real-world processes, a position shared by work done on ADEPT$_{\text{flex}}$ [10] and also our proposed meta-model, where work is allowed to be freely created and delegated by actors, within certain bounds.
A comparison may be drawn between FlexConnect and artifact-centric process modelling [5]. An artifact-centric model explicitly recognises the relationship between data and control flow in a process, and advocates a modularisation of processes around artifacts (essentially business objects). In effect, FlexConnect extends the idea of artifact-centric process modelling to cater for flexible processes.
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 goes beyond the capabilities of DECLARE by including the definition of *creation regions* in which object types (or subtypes) can be instantiated within cardinality restrictions.
A taxonomy of process flexibility by Schonenberg et al. [11] identified and defined four types of flexibility: 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 is flexibility by design, creation is flexibility by deviation and nesting is flexibility by underspecification.
The "Flexibility as a Service" (FAAS) proposal [12] 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 [13],

DECLARE [2] and Worklets [14]. In this paper we have shown how to design flexible process models using OO modelling techniques as an alternative to combining process modelling languages.

Klingemann [15] identified three types of flexible elements in process models: alternative activities, non-vital activities and optional execution order. This framework essentially focuses on flexibility by design. Our framework extends this classification to cater for additional mechanisms such as task delegation and creation regions.

Other object-based process modelling approaches have been proposed by Küster et al. [16] and Wirtz et al. [17]. However, these 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 process modelling is *case handling* [18]. Here, the focus is on the data supporting a system rather than purely on capturing control-flow behaviour. The reasoning behind case handling is that shifting focus away from control-flow leads to less restrictive systems. This view is also supported by Hull et al. [19], Weske et al. [1] and Müller et al. [6] who have proposed process modelling approaches driven by objects and data. Hull et al. and Müller et al. also examine the issue of dynamic changes in data-driven process models. Unlike our approach, the approach of Müller et al. corresponds to "flexibility by change", meaning that the process model is adapted at runtime to deal with unforeseen cases.

In the field of workflow escalation, Georgakopoulos et al. [20] outline an approach to support dynamic changes in workflows in *emergent* situations (e.g. for rescue operations during natural disasters). Their focus is on enabling decision makers to escalate tasks at runtime by changing the course of the workflow execution as required, while retaining some level of control. In contrast, our work focuses on capturing runtime variability of workflows at design-time, instead of escalation.

Some parallels can be drawn between the concept of a COROB, and the *Multiple Instance Without a priori Runtime Knowledge* workflow pattern [21]. Parallels may also be observed between the concept of a ROB and proposals such as Worklets that provide users with a method of dynamically responding to change by taking action not originally envisaged as part of the control-flow behaviour. Our proposal combines these concepts and incorporates them into a process meta-model, which we have expressed in greater detail from our earlier work [23].

7 Summary

In this article 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. In particular we demonstrated how a small set of object types (i.e. Coordination Object, Job 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.

As previously discussed, the main focus of this article is on the design of process models that capture the flexible creation of new objects with the intent of performing unplanned activities at run-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 thresholds to constrain the minimal and maximal number of JOB 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, situations have been encountered that necessitate the definition of *creation regions*. A creation region allows a model designer to establish when instances of a given JOB 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, situations can occur where one needs to constrain the number of JOBs 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 *FlexConnect* modelling tool enables process designers to create O-C process models. A formalisation of the execution semantics for the FlexConnect meta-model is presented as a CPN. To provide object models to the CPN, an export feature was added to FlexConnect that creates an SML file which can then be loaded into the CPN.

This research was supported by an Australian Research Council Linkage Project (LP0562363) co-funded by FlowConnect Pty Ltd.

References

[1] M. Weske, "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 - Track 7. E. Dennis, B. Werner, L. Palagi (Eds.), IEEE Computer Society, 2001.

[2] M. Pesic, M. Schonenberg, N. Sidorova, and W. van der Aalst, "Constraint-Based Workflow Models: Change Made Easy", in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA and IS*, R. Meersman and Z. Tari (Eds.), vol. 4803 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 77–94.

[3] B. Weber, S. Rinderle, and M. Reichert, "Change Patterns and Change Support Features in Process-Aware Information Systems", in *19th International Conference on Advanced Information Systems Engineering*, J. Krogstie, A. Opdahl, G. Sindre (Eds.), vol. 4495 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 574–588.

[4] P. Dadam, M. Reichert, S. Rinderle, M. Jurisch, H. Acker, K. Göser, U. Kreher, and M. Lauer, "Towards Truly Flexible and Adaptive Process-Aware Information Systems", in *Information Systems and e-Business Technologies, 2nd International United Information Systems Conference*, R. Kaschek, C. Kop, C. Steinberger, G. Fliedl (Eds.), vol. 5 of *Lecture Notes in Business Information Processing*, Springer, 2008, pp. 72–83.

[5] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su, "Towards Formal Analysis of Artifact-Centric Business Process Models", in *Business Process Management, 5th International Conference*, G. Alonso, P. Dadam, M. Rosemann (Eds.), vol. 4714 of Lecture Notes in Computer Science, Springer, 2007, pp. 288–304.

[6] D. Müller, M. Reichert, and J. Herbst, "Data-Driven Modeling and Coordination of Large Process Structures", in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA and IS*, R. Meersman and Z. Tari (Eds.), vol. 4803 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 131–149.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley, 1995.

[8] K. Jensen, Coloured Petri Nets. *Basic Concepts, Analysis Methods and Practical Use*. Volume 1. Springer-Verlag, 1997.

[9] S. Rinderle, M. Reichert, and P. Dadam, "Correctness criteria for dynamic changes in workflow systems - a survey", *Data and Knowledge Engineering*, vol. 50, no. 1, pp. 9–34, 2004.

[10] M. Reichert and P. Dadam, "ADEPT$_{\text{flex}}$-Supporting Dynamic Changes of Workflows Without Losing Control", *Journal of Intelligent Information Systems (JIIS)*, vol. 10, no. 2, pp. 93–129, 1998.

[11] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. van der Aalst, “Towards a Taxonomy of Process Flexibility”, in *Proceedings of the CAiSE’08 Forum*, Z. Bellahsene, C. Woo, E. Hunt, X. Franch, R. Coletta (Eds.), vol. 344 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2008, pp. 81–84.
[12] W. van der Aalst, M. Adams, A. ter Hofstede, M. Pesic, and H. Schonenberg, “Flexibility as a Service”, *Database Systems for Advanced Applications, DASFAA 2009 International Workshops: BenchmarX, MCIS, WDPP, PPDA, MBC, PhD*, L. Chen, C. Liu, Q. Liu, K. Deng (Eds.), vol. 5667 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 319-333.
[13] W. van der Aalst and A. ter Hofstede, “YAWL: Yet Another Workflow Language”, *Information Systems*, vol. 30, no. 4, pp. 245–275, 2005.
[14] M. Adams, A. ter Hofstede, D. Edmond, and W. van der Aalst, “Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows”, in *On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA and ODBASE*, R. Meersman and Z. Tari (Eds.), vol. 4275 of *Lecture Notes in Computer Science,* Springer, 2006, pp. 291–308.
[15] J. Klingemann, “Controlled Flexibility in Workflow Management”, in Proceedings of the *12th International Conference on Advanced Information Systems Engineering (CAiSE)*, B. Wangler and L. Bergman (Eds.), vol. 1789 of *Lecture Notes in Computer Science*, Springer, 2000, pp. 126–141.
[16] J. Küster, K. Ryndina, and H. Gall, “Generation of Business Process Models for Object Life Cycle Compliance”, in *Proceedings of the 5th International Conference on Business Process Management (BPM)*, G. Alonso, P. Dadam, M. Rosemann (Eds.), vol. 4714 *of Lecture Notes in Computer Science*, Springer, 2007, pp. 165–181.
[17] G. Wirtz, M. Weske, and H. Giese, “The OCoN Approach to Workflow Modeling in Object-Oriented Systems”, *Information Systems Frontiers*, vol. 3, no. 3, pp. 357–376, 2001.
[18] W. van der Aalst, M. Weske, and D. Grünbauer, “Case handling: a new paradigm for business process support”, *Data and Knowledge Engineering*, vol. 53, no. 2, pp. 129–162, 2005.
[19] R. Hull, F. Llirbat, E. Simon, J. Su, G. Dong, B. Kumar, and G. Zhou, “Declarative workflows that support easy modification and dynamic browsing”, in *Proceedings of the international joint conference on Work Activities Coordination and Collaboration (WACC*), D. Georgakopoulos, W. Prinz, A. Wolf (Eds.), ACM, 1999, pp. 69–78.
[20] D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki, “Managing escalation of collaboration processes in crisis mitigation situations”, in *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, D. Young (Ed.), IEEE Computer Society, 2000, pp. 45–56.
[21] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, “Workflow Patterns”, *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.
[22] G. Redding, M. Dumas, A. H. M. ter Hofstede, and A. Iordachescu, “Generating Business Process Models from Object Behaviour Models”, *Information Systems Management*, vol. 25, no. 4, pp. 319-331, 2008.
[23] G. Redding, M. Dumas, A. H. M. ter Hofstede, and A. Iordachescu, “Modelling Flexible Processes with Business Objects”, In *11th IEEE Conference on Commerce and Enterprise Computing (CEC 2009)*, B. Hofreiter and H. Werthner (Eds.), IEEE Computer Society, 2009, pp. 41-48.

[24] T. Halpin, *Information modeling and relational databases: from conceptual analysis to logical design*. Morgan Kaufmann Publishers Inc., 2001.

[25] M. Schrefl and M. Stumptner, “On the Design of Behavior Consistent Specializations of Object Life Cycles in OBD and UML”, In *Advances in Object-Oriented Data Modeling*, M. Papazoglou, S. Spaccapietra, Z. Tari (Eds.), MIT Press, 2000, pp. 65-104.

[26] M. Schrefl and M. Stumptner, “Behavior-consistent specialization of object life cycles”, *ACM Transactions on Software Engineering Methodology (TOSEM)*, vol. 11, no. 1, pp. 92-148, January 2002.

[27] J. D. Ullman, *Elements of ML Programming*. Prentice-Hall, New Jersey, 1998.