Modelling Flexible Processes with Business Objects

Guy Redding*, Marlon Dumas†*, Arthur H. M. ter Hofstede* and Adrian Iordachescu‡

*Queensland University of Technology, Brisbane, Australia
Email: {g.redding, a.terhofstede}@qut.edu.au

†University of Tartu, Tartu, Estonia
Email: marlon.dumas@ut.ee

‡FlowConnect Pty Ltd, Sydney, Australia
Email: 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: one where processes are organized as interacting business objects rather than as chains of activities. This paper presents a meta-model for business process modelling based on business objects. The paper also presents a real-life case study in which a number of human service delivery processes were designed using the presented meta-model. The case study demonstrates that the meta-model addresses three key flexibility requirements encountered in this domain.

I. 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 flexible workflow models (e.g. [1], [2], [3], [4]).

In this paper we present an approach to capture flexible business processes by centralising the role of objects in process models, in line with recent work on artifact-centric process models [5] and data-driven process modeling [6]. 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 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. In addition, an attempt to impose a standardised way of delivering social services is usually met with resistance by the stakeholders involved in the process – from both providers and consumers.

As a motivating scenario, we consider a possible situation that faces a charity organisation.[1] A recently homeless family makes an application for assistance to a charity. During management of the homelessness case it is discovered that there are other alcoholism and gambling issues that individual family members require assistance with. While each issue can be mapped to a social service that the charity offers, the occurrence of these extra issues is *unplanned*. Unplanned situations are particularly challenging to handle using traditional process modelling notations. They require process models to incorporate several types

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

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

of flexibility, empowering actors with a certain degree of freedom, while enforcing constraints where necessary.

We first present flexibility requirements that were encountered in the context of the above human services case study (Section II). Next, we present a meta-model for capturing flexible processes (Section III). This meta-model is embodied in a tool that allows designers to capture flexible process models and to export these models in a format suitable for simulation and analysis, namely Coloured Petri nets. Next, we show how the proposed meta-model addresses the flexibility requirements (Section IV). The resulting models have been used to realise a system to support these processes in a pilot environment. Finally we compare the proposal with related work (Section V) before concluding and outlining future work (Section VI).

II. Patterns of Flexibility

During the analysis and design of human service delivery processes, we identified recurrent requirements that we grouped 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 uniformity or to enforce certain constraints. We call these patterns of flexibility PoF1-PoF3.

PoF1: Creation Flexibility

Creation flexibility is the ability of a user to trigger the creation of one or more task instances (*jobs*) 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. At the same time, it is sometimes necessary to define constraints regarding the number of task instances and the state(s) in a process from where 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.

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). 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. From a data-flow perspective, a delegatee is a delegator subtype, meaning that the delegatee receives as input data collected by the delegator and produces as output at least the same data as the delegator. Delegation is transitive, meaning that a delegatee may also delegate. This feature, along with the fact that data is transferred from a delegator to a delegatee, distinguishes delegation flexibility from creation flexibility.

PoF3: Nesting Flexibility

Nesting flexibility is the ability of a user to instantiate 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 a drug-related issue which is well beyond the scope of the process that supports 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 the creation of as many ad-hoc sub-process layers as needed to manage issues as they arise, while maintaining sub-process modularity and retaining process control.

III. Flexible Process Meta-Model

We propose to achieve process flexibility via a meta-model, namely *FlexConnect*, that consists of three abstract types of business objects, namely Coordination Object, Job Object and Referral Object (see Figure 1). Concrete business object types inherit from these abstract types.

A **Coordination Object** (COROB) is an object that *coordinates* a process. The COROB is inspired by the recognition that to capture flexible processes, separation must exist between the coordination of tasks and the types of tasks supported by a process. As such, a COROB is responsible for coordination on two levels: (i) creating and synchronising the tasks needed to complete a process; and (ii) referring out-of-scope work to other COROBs.

A **Job Object** (JOB) is an object that represents a job. A JOB manages the execution of tasks involved in a job and reports job completion to its parent object. For

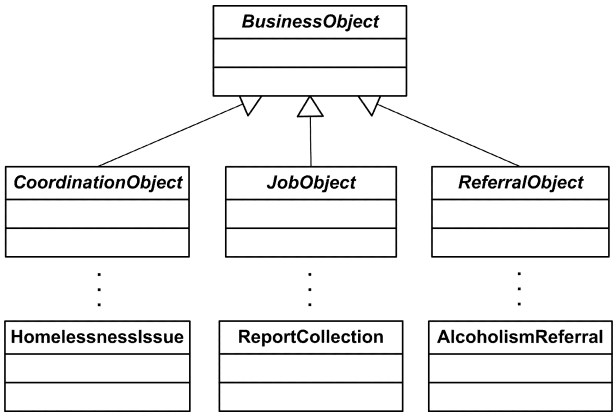

Fig. 1. Abstract Types and Concrete Types

example, two JOBs in the social services process model are the *Report Collection* and *Client Visit* that 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.

A *Flexconnect process model* consists of a set of object types (COROB, JOB and ROB subtypes) and their relations. A meta-model of the object types is shown in Figure 2 as a UML Class Diagram. An object lifecycle is captured as a finite state machine consisting 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 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.

The objects of the proposed framework are represented graphically using the notation in Figure 3. 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.

Communications between objects are modelled using signals. A signal has a label and has a lower-bound and an upper-bound to specify the signal multiplicity, i.e. the minimum and maximum number of times that the signal is sent, where the upper-bound is greater than or equal to the lower-bound. There are two signal subtypes: a *static signal* and a *dynamic signal*.

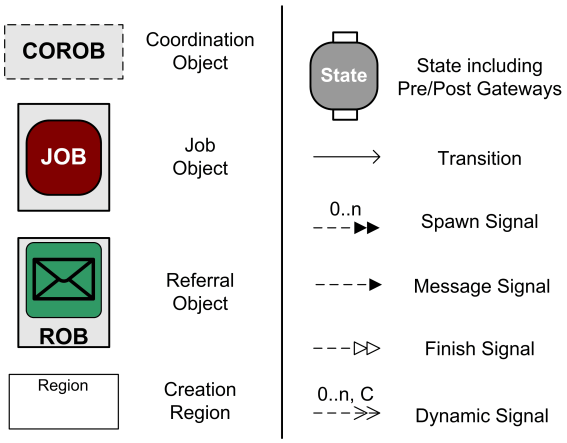

Fig. 3. Modelling Elements

A static signal allows a process designer to model interactions that *will* occur when an object reaches a certain state. The number of times that a signal of a static type can be sent is determined by an expression given at design-time. There are three static signal types: a *spawn signal* to create new object instances, a *finish signal* o indicate object completion, and a *message signal* for object communication following creation and prior to completion. When an object enters a state, it waits for a number of signals attached to its pre-gateway, depending on the *gateway mode* (wait for all, wait for one, or wait until a condition if fulfilled). The object then sits on the state until the activities attached to that state have been completed, and then moves into the post-gateway, where it can produce a number of signals. Thereafter, one of the outgoing transitions is taken based on the conditions attached to these transitions. A state can also be exited "forcefully" if an event attached to one of its outgoing transitions occurs (e.g. a timeout).

In contrast, a dynamic signal allows a process designer to model object communications that *may* occur, i.e. 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 [7].

There are four dynamic signal subtypes: the delegation, creation, referral and nesting signal. A *delegation signal*

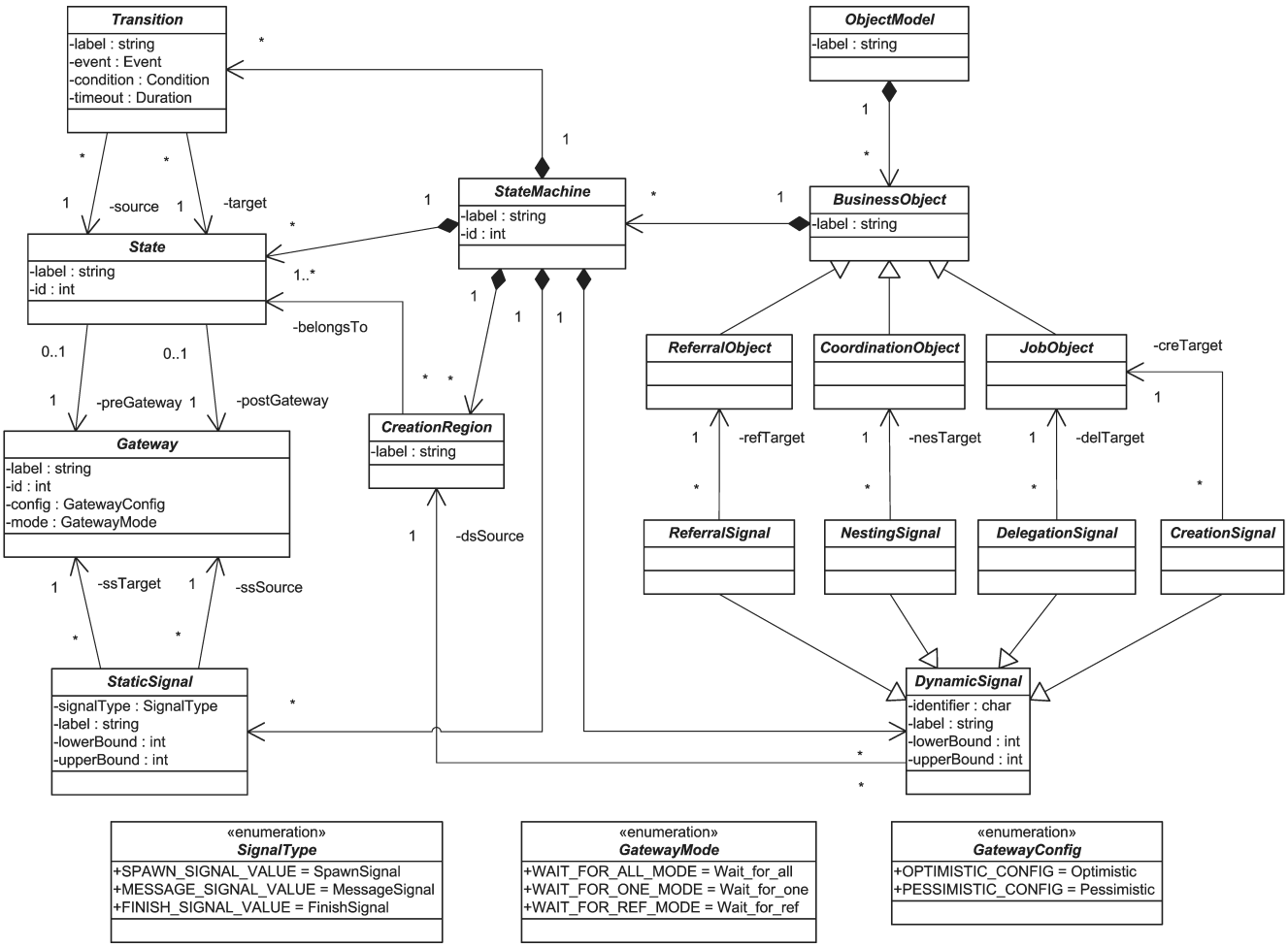

Fig. 2. The FlexConnect Process Meta-Model

allows delegation from a creation region within a delegator JOB to a delegatee JOB. A delegator may delegate to more than one type of delegatee, which must be a subtype of the delegator, but to only one delegatee instance. A *creation signal* enables instances of a JOB to be created from a creation region. The difference between delegation and creation is hereby identified. 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 by the creation signal.

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* to create a ROB if an unexpected major issue arises during the execution of a COROB that is outside the scope of the COROB. 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. The purpose of the ROB is to assist a user in finding a suitable COROB type to address the issue in question.

IV. 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, presented in Figure 5. The model consists of a Client Intake COROB that oversees the process of accepting and evaluating new clients who have contacted the charity for assistance. This COROB is responsible for creating and coordinating the tasks and subprocesses involved in new client intake such as completing a risk assessment, visiting the client and collecting reports from social workers, whilst also coordinating distribution

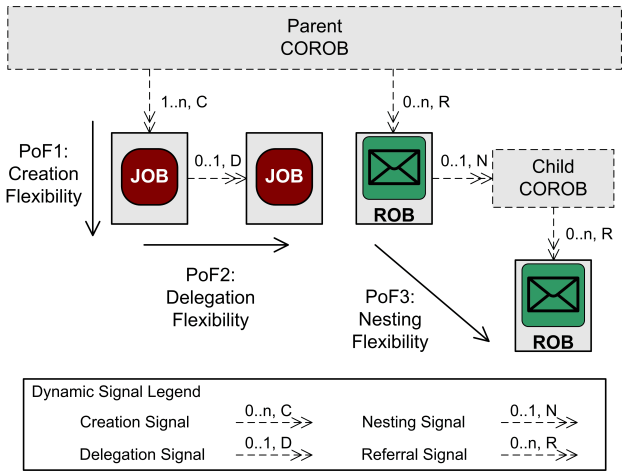

Fig. 4. Patterns of Flexibility

of major issues (if they occur) to other COROBs. To counter the possibility 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 presents extracts from the OO social services process model to show how the patterns of flexibility are supported.

A. Creation Flexibility (PoF1)

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 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 needed (i.e. from the states: "Wait for new plan", "Review plan complete", "Wait for new version" and "Review recorded"), 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 JOB, *0..n* instances of the Child Support JOB and *0..1* instances of the Rental Assistance JOB can be created. This means at least one Client Interaction JOB *will* be created before exiting the Case Management Region, but more instances *may* be created. Any number of Child Support JOBs along with a maximum of one Rental Assistance JOB *may* be created.

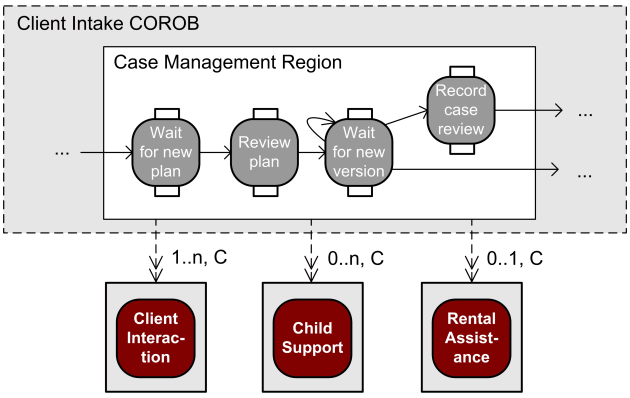

Fig. 6. Creation Pattern of Flexibility

B. Delegation Flexibility (PoF2)

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 four states ("Appointment made", "See client", "Assessment" and "Action approved") and a 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 delegatees from this creation region are the *Skin Treatment*, *Eye Treatment* and *Mental Health Assessment* JOBs which are subtypes of the Client Interaction JOB.

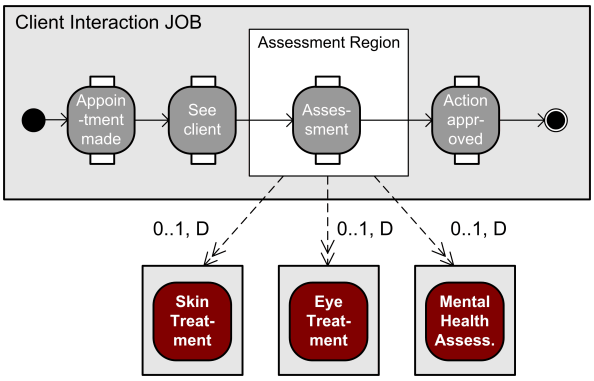

Fig. 7. Delegation Pattern of Flexibility

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 is not allowed if the upper bound is greater than 1 because this implies cloning the delegator.

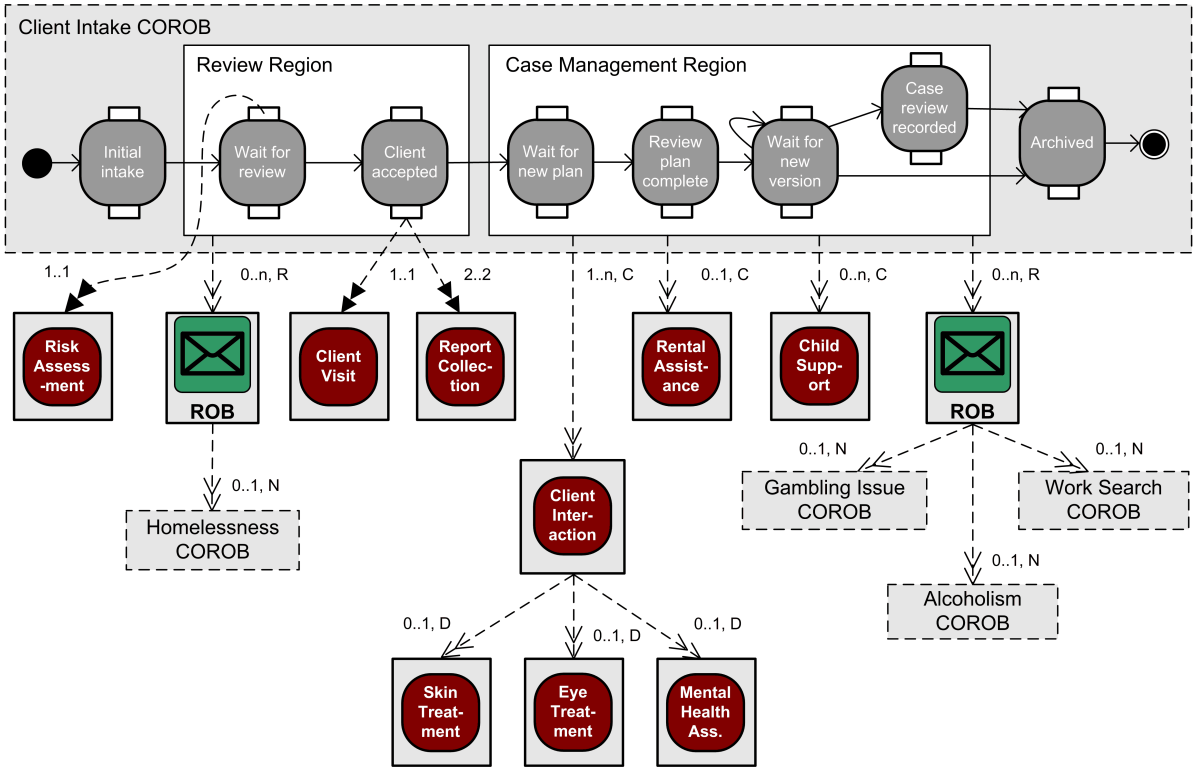

Fig. 5. Object-Oriented Social Services Delivery Model

If multiple instances of a delegator are needed they are firstly created and then permitted to delegate as required. If a delegator does not delegate, then it completes as normal.

C. Nesting Flexibility (PoF3)

Nesting flexibility is motivated by the need to create sub-processes in an ad-hoc fashion. This 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*. If a parent COROB invoke the referral signal an instance of a ROB is created. 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 a parent and child COROB, giving the framework two advantages: (i) COROBs are decoupled, establishing modularity between COROBs, and (ii) the ROB provides the opportunity for human intervention in a referral, since referring major issues in this manner often needs an approval from a third party (e.g. a manager).

In Figure 8 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. If a social worker discovers an alcoholism

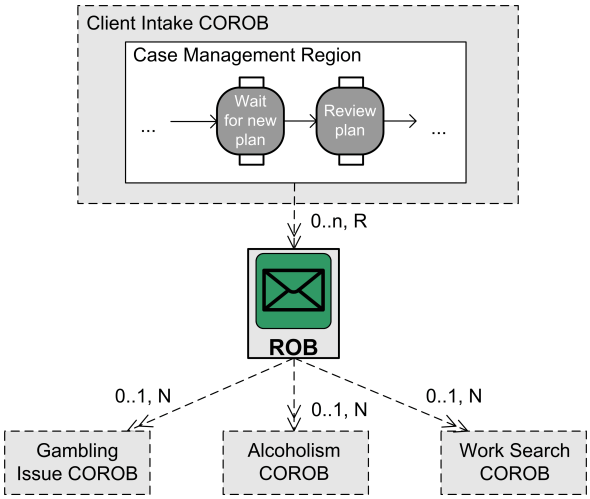

Fig. 8. Nesting Pattern of Flexibility

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

The framework places no restrictions on the levels of

nesting, allowing any number of major issues to be handled and related sub-processes to be created as needed. In the Case Management region an issue resolution plan is prepared for an unemployed client and this unemployment issue is referred through a ROB 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 supports this new problem by referring it to a ROB, which creates a nested Legal Support COROB to manage legal assistance.

D. Putting it all together

Using the examples in this section we have demonstrated how an OO process model coordinates unplanned tasks and issues. The modelling syntax is based on a 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 ways that it may appear is the point of distinction which allows several flexibility requirements that were identified in Section II to be supported, while granting a process model designer the ability to express that flexibility *is* required at particular points and *is not* required at others.

A modelling tool named *FlexConnect*[2] has been developed that allows us to design OO process models as described in this paper. In separate work a formal execution semantics for the proposed meta-model has been defined in terms of a Coloured Petri Net (CPN) [8]. The modelling tool has an export capability to generate a CPN Tools input file that provides an initial marking for the CPN. In addition to providing a formal grounding to the proposal, the CPN provides tool support for analysis of FlexConnect process models. The modelling tool, export function and the generated CPNs have been tested with 20 FlexConnect process models of varying sizes.

V. 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 $\text{ADEPT}_{\text{flex}}$ [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 created 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 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 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

[2]FlexConnect is available at http://code.google.com/p/flexconnect/

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.

Some parallels can be drawn between the concept of a COROB, and the "multiple instance without *a priori* runtime knowledge" workflow pattern [20]. 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.

VI. Conclusions and Future Work

In this paper we demonstrated how the FlexConnect meta-model supports the design of processes consisting largely of unplanned activities. We showed in particular how the three basic business object types of FlexConnect can be combined to capture different patterns of flexibility. The key principle is that business objects specify "what can happen during a case", rather than "how it should happen". Any constraints regarding which objects can be created and when, are overlaid on top of the business 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. 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 number of JOB and ROB objects that can be started by a COROB.

In addition, one may need to define more sophisticated constraints. For example, we have encountered situations that require the definition of "creation regions", that establish *when* instances of a given JOB or ROB type can be created – 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 and type of JOBs or ROBs that need to complete before a COROB moves to a completion state – e.g., a COROB to handle a case for a homeless family will not complete until the range of tasks created to deal with their situation have closed. The definition of such synchronization constraints within the proposed framework is ongoing work.

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, January 3-6 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*, 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*, Trondheim, Norway, June 11-15 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*, Klagenfurt, Austria, April 22-25 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*, Brisbane, Australia, September 24-28 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*, 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$_{flex}$-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*, Montpellier, France, 2008, pp. 81–84.

[12] W. van der Aalst, M. Adams, A. ter Hofstede, M. Pesic, , and H. Schonenberg, "Flexibility as a Service," BPMcenter.org, Tech. Rep. BPM-08-09, 2008.

[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*, Montpellier, France, October 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)*, Stockholm, Sweden, June 5-9 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)*, Brisbane, Australia, September 24-28 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, San Francisco, California, USA*, February 1999, pp. 69–78.

[20] 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.