Modelling Long-Lived Health Care Workflow Transactions

Christopher Meli and George Fernandez*

eHealth Education
c.meli@ehe.edu.au g.fernandez@ehe.edu.au

Abstract. Due to the increasing automation of health care, health care workflows have received significant attention over the last few years. This paper discusses the differences between typical business processes and health care workflows, and introduces a layered architecture suitable for health care workflow models. We present a rules-based approach for modelling long-lived health care workflow transactions, and we discuss a set of transactional integrity rules specific to the workflow patterns found in traditional business processes and show how these rules can be used to design health care workflows with transactional characteristics.

Keywords: health care, workflow, transaction, distributed, business process

1 Introduction

The theory and practice of database transactions is based on the fact that transactions are short-lived. This is because the implementation mechanisms—such as locking or time-stamping—do not work properly if the transaction is not quickly committed to permanent storage and the database made consistent again in a very short time. In contrast, orchestration and choreography of long-lived processes—such as health care and e-commerce flows—often require human or remote systems intervention and may span minutes, hours or days. As these processes are typically distributed in nature, involving multiple services in different organisations, it is not possible to lock records for the required period of time, or time-stamp remote operations, to implement the standard ACID database properties.

Existing methodologies for long-lived transactions extend the traditional transactional model by relaxing the ACID properties, however they do so to a predefined level often making them impractical [12]. Furthermore, these models are considered limited because of their database-centric nature [1].

* Part of this work has been based on unpublished work by Madasamy Madasamy and George Fernandez [13]

In the world of health care there is a need to ensure correct workflow completion, because patients must receive proper complete treatment and all relevant stakeholders must remain informed of any workflow abortions or failures. Thus, in this work we include a discussion of the HCWF elements that need special attention when compared to standard workflows, and propose a set of transactional integrity rules to model long-lived health care workflow transactions.

In particular, this work contributes:

- A layered architectural model applicable for long-lived health care workflow transactions
- A set of integrity rules suitable for modelling long-lived health care workflow transactions

The structure of this paper is as follows:
First in Section 2, we discuss the background of this research, specifically aspects of health care that need to be addressed to make transactions viable for health care workflows and we discuss a potential architecture suitable for our model. Next in Section 3, we present prior related work in the areas of standard and long-lived transactions. In Section 4, we present the necessary definitions and constraints for a workflow model oriented to the area of health care, with transactional characteristics such as commit and abort protocols, backward and forward recovery, and long-lived transactions to implement transactional health care workflows. We discuss the necessary characteristics of the activities and define a set of rules that may be used to validate the transactional consistency of the workflow based on our model. In addition we present a motivating example. In Section 5, we conclude.

2 Background

A standard workflow in its simplest form is defined as "a collection of tasks organised to accomplish some business process" [8]. A workflow specification outlines how such a workflow instance is coordinated, and has a number of different perspectives: `control-flow`, which defines the flow and order of task execution [19]; `data`, which defines the flow of data from one activity to the next [19]; `resource`, which defines the actors responsible for executing these activities [19]; and `operational`, which defines the action an executing activity performs [19].

In distributed workflow management systems the term orchestration is often used to indicate the centralised management of a workflow execution [17]. That is, a single process knows the activity execution schedule and is in control of all workflow activities.

Health Care Workflows (HCWF) are a typical example of process orchestration. They are comprised of many activities such as initial visits, derivation to

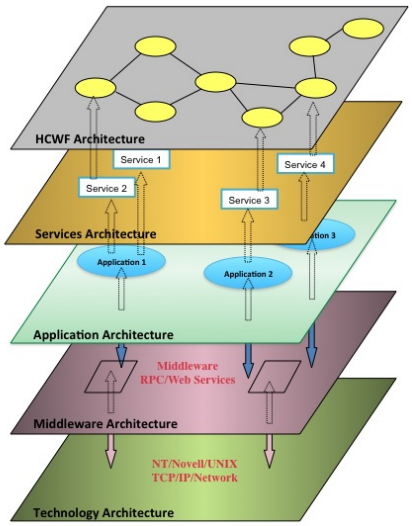

Fig. 1. A Layered Architecture

specialists, requests for analyses and collation of results, conclusion of a health care plan, return visits, etc[1]. As these operations are usually implemented in different systems, HCWF workflows are based on the execution of distributed services housed on heterogeneous software applications implementing these services. Fig. 1 shows a layered architecture appropriate for this model, with the HCWF at the top, and the Services Layer immediately below.[2]

Focusing on the two top layers, there is a separation of concerns between the top layer which focuses on handling a health care case, and the Services Layer which is concerned with the implementation of the workflow as an orchestration of services. The HCWF requires mapping each activity to a composite services orchestration, defining the flow of control between them, and composing them into the workflow and executing it. Naturally, the orchestration must support all standard workflow patterns to provide workflow design capability, correctness and flexibility.

However, the activities of the HCWF are of a different nature and have different characteristics to the ones mapped in the Services Layer. This distinction is crucially important when discussing task characteristics, because although the definitions in Section 4—such as critical and forcible—apply to both layers, the relevant meanings between the two layers is very different.

[1] The literature uses both tasks and activities to indicate each discrete step in a workflow. We use both indistinctly.
[2] Consideration of the other layers is outside the scope of this paper.

So far, health workflows don't seem too different to standard business workflows. However, in distributed workflows there are many points that require special attention, and those relating to the medical domain even further care:

1. At the time of workflow execution starting, some specifics for a given task or service might not be present or known. For instance, the workflow definition may not know the specific laboratory that will conduct the analysis. Hence, the orchestration should support the late binding of tasks to services. This is in contrast to typical business processes such as manufacturing processes where the workflow specification is complete prior to workflow invocation.

2. Health care workflows are often very long and complex, so the orchestration should support nested sub-workflows, and check points to be able to roll-forward or roll-back to that point and then resume the workflow accordingly.

3. In addition to the flow of control, critical activities—the failure of which will abort the workflow—and activities that need to be undone in the event of failure take extra importance, especially at the Services Layer. These failures can be very severe in a medical context when compared to the failure of a standard e-commerce or manufacturing process.

4. The top HCWF layer is very likely to be standard—e.g. imposed by external bodies—regardless of the underlying services architecture, whereas the implementation is fully dependent on this architecture.

5. Compensation of activities, very common in business workflows, is much less common in health workflows (i.e. workflows accompanied by real actions [5]). Again, note that this is only true for the Health Care layer, not for the Services layer where compensation may be a common occurrence. HCWF activity compensation differs, since in most cases, medical actions cannot be undone.

6. In addition, HCWF adherence to all the ACID properties is not appropriate:
   - Each HCWF instance is initialised and associated to a patient, so the underlying systems maintain instance data separately and there is no opportunity for one process to read another processes' 'dirty' data.
   - Similarly, at the top layer each instance executes in isolation from others. The isolation at the lower layers should be guaranteed by the underlying services architecture implementation.
   - Consistency is maintained if the HCWF commits successfully, or else aborts gracefully by cancelling the effects of tasks executed up to the point of the abort.

3 Related Work

The implementation of transactional behaviour in databases has been thoroughly explored [6]. Over the last 20 years transactional behaviour has been success-

fully extended to the field of distributed computing and information systems [16]. Although significant research has been conducted in the area of health care workflows (see for example Browne [4] and Mans [14]), such work does not cover aspects of transactional behaviour.

In the business arena, transactional workflows were originally discussed by Sheth and Rusinkiewicz [18]. Liu et al. [12] present a TWF (Transactional Workflow) model to link transaction support in traditional workflows with the rich business process constructs. The authors argue that the ACID properties are unsuitable for long-lived heterogeneous distributed workflow processes. Thus, they relax the strict ACID properties by defining a set of task properties: `Critical`, `Compensatable`, `Forcible`, and `Undo-Not-Required`. These properties allow workflow designers to relax the strict transactional behaviour whilst still providing a certain level of transaction support.

In addition to these properties, the authors propose a set of complex task composition patterns: `Alternative`, `Contingency`, `Conditional`, and `Iterative` and also include the ability for nested transactional workflows (Hierarchical Workflows). These patterns act as control flow constructs and coordination constraints. Each of the patterns, including the top level workflow is represented as a finite state machine. The finite state machine of each task is combined into an overall transactional workflow and forms the execution model. The authors are then able to define a set of mappings between the state machines for each task and the 'visible state' of the overall TWF. They also present a TWF scheduling and execution architecture to determine the workflows correctness and ability to reach an acceptable termination state.

Alonso et al. [1] compare and contrast the advanced transactional models with the needs of business processes and workflow management systems, and present a focused analysis of two transactional models [7, 5] that can be implemented to support transactional workflows.

Garcia-Molina and Salem [7] propose the concept of Sagas, long-lived transactions that are composed of individual sub-transactions that each perform independent items of work and can be interleaved with other transactions. In addition to the primary flow sub-transactions, each sub-transaction has an associated compensation sub-transaction. In the normal case, each sub-transaction can commit once executed, preventing database locking for long periods. Assuming all transactions are able to commit the saga has committed successfully. In the event that a sub-traction fails, the saga must be aborted, and the compensation sub-transactions for all executed primary flow transactions are executed in reverse order, undoing the sagas effects. In this way, a saga provides long-lived atomicity without long-lived database locking.

Elmagarmid et al. [5] propose an advanced transaction model to support the execution of global long-lived transactions in a multi-database system (a collection of distributed, heterogeneous, independent database systems). The model takes a multi-directional approach. First it uses the fact that there can

be multiple databases that can execute a sub-transaction. Second, the authors take advantage of the fact that some sub-transactions can be semantically "undone", which allows these sub-transactions to be committed early, before the non-compensable transactions. Associating compensable sub-transactions was proposed by Gray et al. [9], and is used in the saga model in [7]. However, the authors expand on this by mixing compensatable sub-transactions with non-compensatable sub-transactions to help reduce the isolation granularity of the global transaction. Thirdly, the authors' approach makes use of temporal scheduling.

Li et al. [11, 10] propose a calculus based formalism for composing long-lived transactions. The authors' language, called t-calculus, uses algebraic semantics to define long running transactions that support weak atomicity by activating compensation flows in the event of activity failures.

In Bhiri et al. [2], the authors propose an approach for composing Transactional Composite (Web) Services (TCS). First, composite service designers select the required component services that will form the TCS, and arrange the flow between these component services using workflow patterns (see [19] for example patterns). Then, designers associate transactional dependencies between component services to form the transactional flow. Next, designers define a set of Acceptable Termination States for the component services, which in turn defines the TCS' failure atomicity. The approach then involves using a set of transactional validity rules to compute a set of transactional properties for the composite services, which can then be verified to ensure the TCS is transactionally consistent.

The patterns suggested in Bhiri et al. [3] combine workflow patterns together with transactional dependencies to form transactional patterns that can be connected together, suggesting an approach for reliable web service composition. This work provides two approaches to compose TCS's and ensure transactional consistency: The first approach is to compose a TCS by connecting the required transactional patterns together and assigning the desired transactional dependencies. Their matching algorithm is then able to match the patterns and transactional dependencies with suitable web-services that have the required transactional properties; The authors second approach is to connect the required transactional patterns together with the desired web-services, and then the authors algorithm will use the potential web-service transactional properties to infer and assign required transactional dependencies. However, their discussion does not include the application of transactional reliability to complex workflow patterns like multi-choice, multi-merge, synchronising merge, and arbitrary cycles.

In refining workflow models to support long-lived transactions these works demonstrate that there have been significant efforts to provide finer activity semantics, including characterising activities and their impact on the successful completion of a workflow. They either address a particular aspect of long-lived

transactions, propose an execution model, present an approach for transactional web-service composition but do not consider some characteristics of transactional workflows or are relevant health care workflow research. However, in this work we investigate the rules necessary for transactional workflow composition, whilst considering the semantics of the target domain—health care in this work—and the relevant transactional characteristics discussed in some research works above.

4 Long-Lived Healthcare Transactions

Before discussing the proposed long-lived transaction integrity rules, we first briefly discuss some of the semantics and dependency rules proposed in existing literature, that we make use of in our proposed rules.

In Liu et al. [12], the authors proposed several task properties that characterise the tasks semantics. These are: `Forcible`, `Critical`, `Compensatable` and `Undo-not-required`. Forcible is defined as a task guaranteeing to eventually succeed. A critical task is one that must have committed successfully if the transactional workflow commits, whereas non-critical tasks may have aborted. A compensatable task is one that can be undone in the event the workflow is aborted. Finally an undo-not-required task does not need to be reversed if the task commits but the transaction is aborted, (see [12] for further details).

In addition to the above semantics, the authors also combine two task dependencies identified in Bhiri et al. [3]. These are the `Activation` dependency, which is the activation of task T2 after task T1 has been executed, and the `Alternative` dependency, which refers to a contingency task that is activated should the primary task fail, (see [3] for further details).

4.1 Transactional Integrity Rules for HCWF Composition and Orchestration

In this section we introduce our proposed transactional integrity rules, that ensure a workflow maintains the ability to either commit successfully or satisfactorily abort if required. For each of the common workflow patterns defined in van Der Aalst et al. [19]: `sequence`, `parallel split`, `exclusive choice`, `simple merge`, `multi-choice`, `synchronising merge`, and `arbitrary cycles`, we define the rules and constraints needed to ensure correct transactional behaviour. The patterns discussed are numbered to coincide with those used in [19]. For each of the patterns, we provide a brief description of the patterns behaviour and direct the reader to [19] for further details.

In the following discussion, tasks may be designated as Dependent or Independent. Dependent tasks are those that have a dependency attached. We say that task T1 is a source if it has attached another task T2. Task T2 is then called the contingency task. Independent tasks are those without a dependency attached.

Transactional Workflow Commit Protocol

Once the workflow designer has composed the transactional workflow (TWF) using the rules discussed in the patterns below, the design has to be validated for any violations of the following guidelines. If there is a violation then the workflow needs to be altered.

1. A TWF can commit if all critical tasks in the workflow have committed.
2. A TWF can abort if all the undo-required tasks are compensated.
3. A TWF cannot abort if a critical non-compensatable task has been executed.
4. A TWF if all the undo-required tasks are compensatable.
5. A TWF is said to be forcible if all the critical tasks in the workflow are forcible. A forcible workflow is always said to commit.

Pattern 1 (Sequence)

Description: Two activities `A` and `B` are in sequence if the completion of activity `A` enables activity `B` for execution.

Transaction Rules/Constraints:

1. A set of critical tasks can be executed sequentially without being compensatable if all those tasks are forcible.

2. A set of critical and non-critical tasks can be executed sequentially if all critical tasks are forcible.

3. A set of critical tasks can be executed sequentially without any forcibility constraints if all those tasks are compensatable and none of the tasks is a source of an alternative dependency to a critical, non-compensatable task.

4. For a set of critical compensatable tasks with one or more tasks being the source of alternative dependency to a critical non-compensatable task, then let task T1 be the first critical compensatable task having an non-compensatable task as the contingency option in the sequential pattern:
   (a) All independent critical tasks following task T1 are forcible.
   (b) For each dependent task following task T1, the contingent task of the alternative dependency is forcible.

5. A set of critical tasks and non-critical tasks can be executed sequentially without any forcibility constraints if the set of critical tasks are compensatable.

6. If there is a critical non-compensatable task in the workflow, then the workflow is transactionally consistent if all subsequent critical tasks following the critical non-compensatable task are forcible.

7. In a workflow or sub-workflow, a critical non-compensatable task T2 can be executed after a critical non-compensatable task T1, if:
   (a) T2 is an independent critical non-compensatable task, then T2 has to be forcible.
   (b) T2 has an alternative dependency to a task T3, then T3 has to be forcible.
   (c) All subsequent critical tasks in the current sub-workflow and the following sub-workflows are forcible.

Pattern 2 (Parallel Split / Concurrency)

Description: A split from a single execution path into multiple concurrently executing paths.

Transaction Rules/Constraints:

1. A set of critical compensatable tasks can be executed concurrently if non of the tasks in the set is a source of alternative dependency to a critical non-compensatable task.

2. A set of critical compensatable tasks with at least one task in the set being the source of an alternative dependency to a critical non-compensatable task, can be executed concurrently if:
   (a) All the critical compensatable tasks which are not a source of alternative dependency are forcible.
   (b) In the case of more than one contingent non-compensatable task in the workflow, then all the contingent non-compensatable tasks are forcible.

3. A set of tasks that are not a source of alternative dependency can be executed concurrently if:
   (a) All the critical non-compensatable tasks are forcible, and
   (b) All the subsequent critical tasks in the workflow are forcible.

4. A set of critical non-compensatable tasks with at least one task in the set being the source of alternative dependency to a critical non-compensatable task, can be executed concurrently if:
   (a) All the concurrently executing critical non-compensatable tasks which are not a source of alternative dependency are forcible.
   (b) All the critical tasks which are the contingent tasks of the alternative dependency are forcible.
   (c) All the subsequent critical tasks in the workflow are forcible.

5. A set of critical compensatable tasks can be executed concurrently with a set of critical non-compensatable tasks if:
   (a) All the critical compensatable tasks that are not a source of alternative dependency are forcible.

(b) All the critical non-compensatable tasks that are not a source of alternative dependency are forcible.
(c) For every task in the set that is the source of an alternative dependency, the contingent task of the alternative dependency is forcible.
(d) All the subsequent critical tasks in the workflow are forcible.

6. A set of critical non-compensatable tasks can be executed concurrently with the set of non-critical compensatable tasks and/or non-critical non-compensatable tasks only if:
   (a) All the concurrently executing critical non-compensatable tasks which are not a source of alternative dependency are forcible.
   (b) For every task in the set which is the source of an alternative dependency, the critical contingent task of the alternative dependency is forcible.
   (c) All the subsequent critical tasks in the workflow are forcible.

   As the failure of the non critical tasks does not require the workflow to abort, they need not be forcible.

7. A set of critical tasks and non-critical tasks can be executed concurrently without any forcibility constraints if all the critical tasks executing concurrently are compensatable. If there exists at-least one critical task that is a source of an alternative dependency to a critical non-compensatable task then rule 2 shall be satisfied.

Pattern 4 (Exclusive choice)

Description: A point in the execution path where a decision is made to select one of many alternative execution paths.

Transaction Rules/Constraints:

1. A set of independent compensatable tasks can be used in the exclusive choice pattern of the workflow without any forcibility constraint imposed.

2. A set of compensatable tasks with one or more tasks being the source of alternative dependency to a critical non compensatable task, can be used in the exclusive choice pattern of the workflow if all the subsequent critical tasks in the workflow are forcible.

3. A set of compensatable and non compensatable tasks can be used in the exclusive choice pattern of a workflow if all the subsequent critical tasks in the workflow are forcible.

4. A set of critical non-compensatable tasks with one or more tasks being the source of alternative dependency to a critical non-compensatable task, can be used in the exclusive choice pattern of the workflow if the contingency task and all the subsequent critical tasks in the transaction are forcible.

Pattern 5 (Simple merge)

Description: A point that takes many execution paths, where only one path is active and merges them back into a single execution path.

Transaction Rules/Constraints:

1. If one of the incoming branches of the simple merge has a non-compensatable task associated with it, then:
   (a) All the tasks in every incoming branch that is being merged shall be made forcible.
   (b) All subsequent critical tasks following the merge point shall be made forcible.

Pattern 6 (Multi-choice)

Description: A point where a single execution path is split into many paths where any number of paths may be concurrently executed.

Transaction Rules/Constraints:

1. If an OR-split or selection pattern of a workflow has a set of compensatable tasks with one or more tasks being the source of alternative dependency to a critical non-compensatable task then:
   (a) All the independent tasks specified in the OR-split pattern shall be forcible.
   (b) All the contingent tasks specified in the OR-split shall be forcible.
   (c) All the subsequent critical tasks in the workflow following the OR-split shall be forcible.

2. If an OR-split or selection pattern of a workflow has more than one non compensatable task, then:
   (a) All the independent tasks specified in the OR-split pattern shall be forcible.
   (b) All the contingent tasks specified in the OR-split pattern shall be forcible.
   (c) All the subsequent critical tasks in the workflow following the OR-split shall be forcible.

Pattern 7 (Synchronising merge)

Description: A point that merges multiple execution paths where only some may be concurrent executing into a single execution path, waiting for all active incoming act branches before continuing execution.

Transaction Rules/Constraints:

1. If one of the incoming branches of the synchronising merge has a non-compensatable task associated with it, then:
   (a) All the tasks in every incoming branch that is being merged shall be forcible.
   (b) All subsequent critical tasks following the merge point shall be forcible.

Pattern 10 (Arbitrary cycles)

Description: A set of activities that form a loop and are executed repeatedly.

Transaction Rules/Constraints:

1. A set of independent compensatable and non-compensatable tasks can be used in the cycle of a workflow if:
   (a) All the tasks involved in the cycle are forcible irrespective of compensatability.
   (b) All the subsequent critical tasks in the current sub-workflow and following sub-workflows are forcible.

2. A set of dependent compensatable and non-compensatable tasks can be used in the cycle of a workflow if:
   (a) All the independent tasks involved in the cycle are forcible irrespective of compensatability.
   (b) All the contingent tasks involved in the cycle are forcible.
   (c) All the subsequent critical tasks in the current sub-workflow and following sub-workflows are forcible.

4.2 Discussion and Motivating Example

We now consider a sub-set of a long-lived stroke management health care workflow that has been translated[3] from the recommendations provided in the Clinical Guidelines for Stroke Management 2010 (see NSF [15]), and have adapted into a HCWF process, see Fig. 2.

For each of the tasks in Fig. 2, Table. 1 defines the associated semantics for each task. For reasons of page count limitations, the authors select a few of the most common transactional integrity rules and discuss there use in the motivating example shown in Fig. 2.

We can see from the first three tasks (Detailed History), (Clinical Examination) and ($ABCD^2$ Score), that they are connected sequentially [Pattern 1] and therefore each have an activation dependency with the prior task. In Table. 1, the (Detailed History) task is has no semantics associated with it. In some cases, a patient may have no existing history (first emergency visit) and may not physically be capable of providing existing history. Thus, the task cannot be forcible. Given that the patients history may have the least impact in this treatment process, we can say the task is non-critical. The two tasks (Clinical Examination) and ($ABCD^2$ Score) on the other hand are designated forcible and critical. For the (Clinical Examination), a doctor can always perform a patients examination, making it forcible, and without the task completing successfully,

[3] Translation required subjective interpretation of the relevant guideline recommendations.

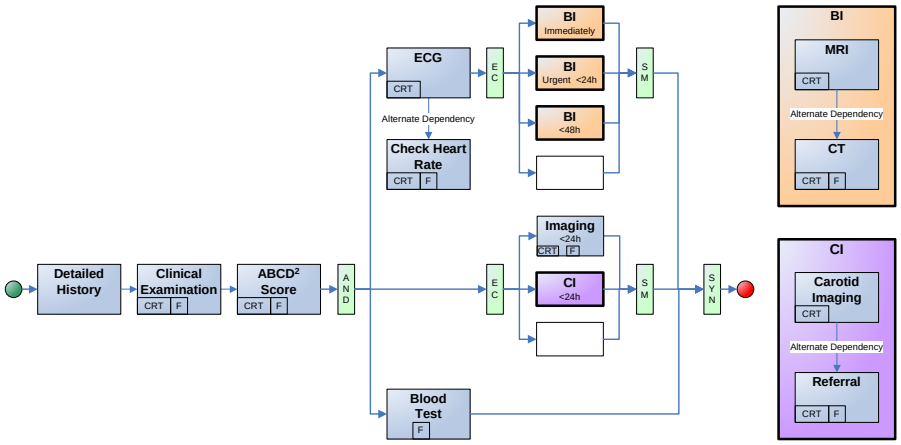

Fig. 2. Health Care Workflow for stroke management assessment phase

Table 1. Health Care Workflow transactional characteristics

	Forcible	Critical	Compensatable	Alternative Dependency
Detailed History				
Clinical Examination	✓	✓		
$ABCD^2$ Score	✓	✓		
MRI		✓		CT
CT	✓	✓		
ECG		✓		Heart Rate
Heart Rate	✓	✓		
Carotid Imaging		✓		Referral
Referral	✓	✓		
Blood Test	✓			

the process cannot proceed and commit, making it critical. Similarly for task ($ABCD^2$ Score), the $ABCD^2$ Score can always be calculated and the result is required upstream so it too is forcible and critical. According to sequential rule two, all three tasks can form a transaction and be executed successfully as we are combining a non-critical non-forcible task (Detailed History) with two critical tasks (Clinical Examination) and ($ABCD^2$ Score) that are both forcible.

Next we look at the exclusive choice and simple merge patterns. Towards the top centre we have an exclusive choice leading into three instances of the brain imaging (BI) task that are merged back together using a simple merge. The three instances are used to separate the different timing constraint choices available for the Brain Imaging activity. The activity itself is a compound activity identified by the thicker boarder. On the right, we can see the BI activity is composed of two tasks, an (MRI) scan that is critical and an alternative of a (CT) scan that is also critical as well as forcible. Having a critical non-compensatable task

as a source of an alternative dependency fits into rule 4 of the exclusive choice pattern above. The (CT) scan task is required to be forcible in the event the (MRI) task cannot be successfully completed. The exclusive choice pattern is again used in the centre for the carotid imaging (Imaging) and (CI) activity.

The two exclusive choice patterns can both be considered activities within the concurrency pattern (AND split and Synchronising join). Following on from the ($ABCD^2$ Score) task, the single execution path splits into three concurrently executing paths. In the first path a critical non-compensatable (ECG) task is followed by the brain imaging exclusive choices pattern. In the second path is the carotid imaging exclusive choice pattern. In the third path we have the (Blood Test) task. Since each of the exclusive choice transactions will eventually commit, and given (ECG) is critical and a source dependency to (Check Heart Rate), and task (Blood Test) is forcible, concurrency rule 4 is satisfied.

Thus in this way, transactionally correct workflows may be composed and executed. It should also be mentioned that each transaction can act as a checkpoint, such that a transaction abortion does not require the whole workflow to be rolled-back, the patient can continue from their last checkpoint.

5 Conclusion

In this work, we introduced long-lived health care workflows and presented an applicable multi-layered architecture that encompasses both health care workflow and distributed services aspects. In doing so, we also outlined the differentiating factors of normal business processes and health care workflows.

Following on we present the core of this work, our rules-based approach to modelling long-lived HCWF transactions. The approach ensures HCWF transactions are transactionally correct and will either commit successfully or abort gracefully. Further research in this area will focus on a deriving further rules, a BPEL engine implementation and a mathematical formalisation.

References

[1] Gustavo Alonso, Divyakant Agrawal, Amr El Abbadi, Mohan Kamath, Roger Gunthor, and C Mohan. Advanced transaction models in workflow contexts. In Data Engineering, 1996. Proceedings of the Twelfth International Conference on, pages 574–581. IEEE, 1996.

[2] Sami Bhiri, Olivier Perrin, and Claude Godart. Ensuring required failure atomicity of composite web services. In Proceedings of the 14th international conference on World Wide Web, pages 138–147. ACM, 2005.

[3] Sami Bhiri, Olivier Perrin, Claude Godart, et al. Extending workflow patterns with transactional dependencies to define reliable composite web services. In Advanced International Conference on Telecommunications and

International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), pages 145–150, 2006.

[4] Eric Donald Browne. Workflow modelling of coordinated inter-health-provider care plans. PhD thesis, University of South Australia, 2005.

[5] Ahmed Elmagarmid, Yungho Leu, Witold Litwin, and Marek Rusinkiewicz. A multidatabase transaction model for interbase. Technical report, Purdue University, 1990.

[6] Ahmed K Elmagarmid. Database transaction models for advanced applications. Morgan Kaufmann Publishers Inc., 1992.

[7] Hector Garcia-Molina and Kenneth Salem. Sagas. In Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, SIGMOD '87, pages 249–259. ACM, 1987.

[8] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and Parallel Databases, 3(2):119–153, 1995.

[9] Jim Gray et al. The transaction concept: Virtues and limitations. In VLDB, volume 81, pages 144–154, 1981.

[10] Jing Li, Huibiao Zhu, and Jifeng He. Algebraic semantics for compensable transactions. In Theoretical Aspects of Computing–ICTAC 2007, pages 306–321. Springer, 2007.

[11] Jing Li, Huibiao Zhu, Geguang Pu, and Jifeng He. Looking into compensable transactions. In Software Engineering Workshop, 2007. SEW 2007. 31st IEEE, pages 154–166. IEEE, 2007.

[12] Chengfei Liu, Dean Kuo, Michael Lawley, and Maria E Orlowska. Modeling and Scheduling of Transactional Workflows. pages 1–13, 1996.

[13] Madasamy Madasamy and George Fernandez. Transactional Enterprise Workow Patterns [A Rule-Based Approach]. Personal Communication, March 2014.

[14] R Mans. Workflow support for the healthcare domain. PhD thesis, Eindhoven University of Technology, 2011.

[15] National Stroke Foundation (NSF). Clinical guidelines for stroke management 2010, 2010. URL `http://strokefoundation.com.au/site/media/clinical_guidelines_stroke_managment_2010_interactive.pdf`.

[16] M Tamer Özsu and Patrick Valduriez. Principles of distributed database systems. Springer, 2011.

[17] Stephen Ross-Talbot. Orchestration and choreography: Standards, tools and technologies for distributed workflows. In NETTAB Workshop-Workflows management: new abilities for the biological information overflow, Naples, Italy, 2005.

[18] Amit Sheth and Marek Rusinkiewicz. On transactional workflows. Data Engineering Bulletin, 16(2):20, 1993.

[19] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. Distributed and parallel databases, 14(1):5–51, 2003.