MEDINFO 2019: Health and Wellbeing e-Networks for All
L. Ohno-Machado and B. Séroussi (Eds.)

doi:10.3233/SHTI190345

Providing Comorbid Decision Support via the Integration of Clinical Practice Guidelines at Execution-Time by Leveraging Medical Linked Open Datasets

William Van Woensel[a], Samina Abidi[a], Borna Jafarpour[a], Syed Sibte Raza Abidi[a]

[a] *NICHE Research Group, Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada,*

Abstract

Clinical Decision Support Systems (CDSS) utilize computerized Clinical Practice Guidelines (CPG) to deliver evidence-based care recommendations. However, when dealing with comorbidity (i.e., patients with multiple conditions), disease-specific CPG often interact in adverse ways (e.g., drug-drug, drug-disease interactions), and may involve redundant elements as well (e.g., repeated care tasks). To avoid adverse interactions and optimize care, current options involve the static, a priori integration of comorbid CPG by replacing or removing therapeutic tasks. Nevertheless, many aspects are relevant to a clinically safe and efficient integration, and these may change over time—task delays, test outcomes, and health profiles—which are not taken into account by static integrations. Moreover, in case of comorbidity, clinical practice often demands nuanced solutions, based on current health profiles. We propose an execution-time approach to safely and efficiently cope with comorbid conditions, leveraging knowledge from medical Linked Open Datasets to aid during CIG integration.

Keywords:

Practice Guideline, Decision Support Systems, Clinical; Comorbidity

Introduction

Clinical Practice Guidelines (CPG) are disease-specific and evidence-based recommendations to guide diagnosis, prognosis, and treatment [1]. By computerizing CPG as *Computer Interpretable Guidelines (CIG)*, they can be utilized by *Clinical Decision Support Systems (CDSS)* to deliver evidence-informed care recommendations. While CDSS can effectively manage a single, complex condition, they often cannot cope with multiple illnesses in a single patient (i.e., comorbidity). Indeed, managing comorbid conditions in a patient is a complex problem, and requires avoiding adverse interactions (e.g., drug-drug or drug-disease), by tailoring or adding new therapeutic interventions; and optimizing care and resource utilization, by eliminating redundant investigations. Hence, the direct, concurrent application of multiple disease-specific CIG is considered neither a safe nor efficient option.

A possible approach for managing comorbidity involves reviewing existing, disease-specific CPG, and manually integrating their workflows to yield a single, clinically safe and resource-optimized clinical workflow. Examples include the England and Wales NICE (National Institute for Health and Care Excellence) CPG [2], which manages kidney disease combined with hypertension, lipids and hepatitis C; and Abidi [3] performed a manual, ontology-based integration of CIG for Congestive Heart Failure (CHF) and Atrial Fibrillation (AF). Nonetheless, this solution is considered impractical, since it is not feasible to apply it to every possible comorbidity. In line with his, a range of computer-based approaches have been proposed to semi-automatically integrate comorbid CIG [3-7]. These approaches tend to focus on the a priori integration of comorbid CIG to realize a single, static clinical workflow. Nevertheless, we observe that execution-time aspects driving *clinical workflow execution*—i.e., lab test outcomes, medical resource availability, care delays, and the dynamic patient's health profile—also affect the clinical safety and efficiency of *comorbid CIG integration*. Due to runtime delays, test outcomes or changes in the patient's health parameters, integration decisions may need to be applied, adapted or reverted. Hence, we argue that a comorbid CIG integration approach should incorporate the *execution-time flexibility to dynamically apply or revise previous integration decisions*, in light of evolving clinical profiles and operational aspects. Below, we introduce 3 running examples (based on illness-specific CPG) which will be used to illustrate our work:

- *OA-Diabetes comorbidity*: both osteoarthritis (OA) and diabetes guidelines recommend Non-Steroidal Anti-Inflammatory Drugs (NSAID), such as aspirin and ibuprofen. Since NSAID increase the risk of bleeding in OA patients, this type of drug should only be described once for a comorbid patient.

- *AF-CHF / COPD-PE*: many illnesses require imaging scans, ECG and other tests (e.g., spirometry) at a point-of-care or healthcare facility. With many of these illnesses often occurring comorbidly, such as AF-CHF and Chronic Obstructive Pulmonary Disease (COPD) – Pulmonary Embolism (PE), grouping these tests into a single visit will improve the patient's quality of life and reduce incurred costs.

- *HTN-Diabetes*: thiazide diuretics such as Methyclothiazide are often prescribed for treating hypertension (HTN). However, they may worsen glycemic control for diabetes patients. In regular cases, the dose of thiazide diuretics should be reduced; when creatinine clearance is low and volume control is needed, it should be replaced by loop diuretics.

We present a knowledge-driven, execution-time approach for integrating comorbid CIG into a clinically safe and efficient comorbid clinical workflow. To cope with execution-time aspects and clinical pragmatics, our approach supports (1) refining or reverting integration decisions as more data becomes available—such the outcomes of medical tests, unforeseeable runtime delays, or limited hospital resources; and (2) defining alternative CIG integration decisions, which are conditional on the patient's health profile and other clinical events. Our approach involves a *CIG Integration Framework*, which employs a Semantic Web based approach to CIG representation, integration and execution. To support the identification of adverse interactions, we leverage various Linked Open Datasets (LOD), including the SNOMED CT ontology[1], the Drug-Drug Interactions Ontology (DINTO)[2], the Drug Ontology (DrOn)[3] and Bio2RDF DrugBank ontology[4].

Methods

Our approach supports a practitioner in (a) identifying the set of clinical tasks, called *CIG integration points*, which could result in loss of clinical safety or efficiency in comorbid sitations; (b) given these integration points, instantiating appropriate *CIG integration policies* that ensure clinical safety and efficiency of the integrated CIG. In particular, a *CIG integration policy* is instantiated to deal with a particular comorbid patient scenario: e.g., to cope with drug interactions in a HTN-Diabetes comorbidity, a policy is instantiated to adjust dosages of Methyclothiazide.

The CIG Integration Ontology (*CIG-IntO*) defines the core terms for representing, integrating and executing CIG. We note that the SNOMED CT taxonomy is integrated into our CIG-IntO ontology, with an extended mapping to other ontologies (DrOn, DINTO, DrugBank). As we will illustrate in the paper, by leveraging knowledge from these datasets, our CIG framework can aid the practitioner in discovering CIG integration points.

A CIG integration policy is responsible for coordinating the execution-time integration of comorbid CIG; which is done by *operating on a multi-level state machine*. At runtime, these state machines manage the lifecycles for integration points. We elaborate on these lifecycles, and the available integration operations, below. Next, we discuss the high-level semantics of our proposed CIG integration policies.

Clinical task lifecycles

Workflow-related task lifecycle

Fig. 1 shows the workflow-related clinical task lifecycle.

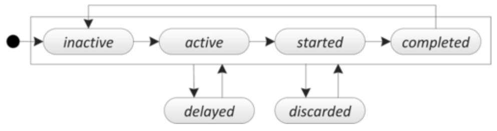


Figure 1– State machine for the CIG task workflow lifecycle.

During regular execution, a CIG task travels from *inactive* to *active* when it is next in line for execution. This may occur when its associated pre-conditions are satisfied and/or the previous task was completed. The practitioner selects an *activated* task for execution at their discretion, thus moving it to the *started* state. Finally, an *activated* task moves to the *completed* state when a practitioner marks it as complete, or when indicated by an external system (e.g., lab information system). Below, we exemplify the *CIG execution semantics* [8] (using OWL2 DL [9]) that move *inactive* tasks to the *active* state:

$$InactiveTask \cap TaskWithSatisfiedCondition \cap \exists hasPrevious.CompletedTask \subset \exists hasNewTaskState.Active \quad \textbf{(1)}$$

An *InactiveTask* with a satisfied pre-condition (*TaskWithSatisfiedCondition* class) and a prior, *completed* workflow task (*hasPrevious* relation), will be assigned the new *Active* state.

[1] http://browser.ihtsdotools.org/
[2] https://bioportal.bioontology.org/ontologies/DINTO
[3] https://bioportal.bioontology.org/ontologies/DRON
[4] http://bio2rdf.org/

To coordinate the integration of comorbid CIG, an integration policy can apply *integration operations*, which move a clinical task to the *delayed* or *discarded* state, or *replace* a task in the CIG workflow. To ensure continuing clinical safety and efficiency in light of changing execution-time circumstances, our approach supports *reverting* these operations as well.

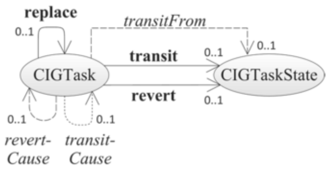


Figure 2– Integration operations on clinical tasks (CIG-IntO).

The *transit* relation indicates a transit operation on a CIG task (*CIGTask*) to a particular state (*CIGTaskState*). To support reverting the operation, the *transitFrom* property keeps the original *from* state. The *revert* operation is represented by the *revert* relation, indicating the state that the CIG task should be reverted to. The *replace* relation indicates the clinical task (subject) to be replaced and the replacement task (object). The causes for *transit* and *revert* operations are kept as well.

Outcome-related task lifecycle

Various outcome-related information, including medical test results or physician/patient choices, will only become available as the clinical workflow progresses. When faced with such new execution-time information, additional integration operations may be needed, while previous operations may need to be adapted or reverted. To that end, CIG tasks additionally feature an outcome-related lifecycle:

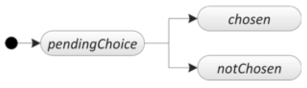


Figure 3– Integration operations on clinical tasks.

A clinical task is in the *pendingChoice* state when no choice has yet been made at the nearest preceding decision node; i.e., it is still unknown whether the task will be executed. After a choice is made, all tasks in the "chosen" branch travel to the *chosen* state, up until the next decision node; and tasks in the non-chosen branch(es) travel to the *notChosen* state. In other words, tasks in the *chosen* state are in line for execution, while *notChosen* tasks will not be executed.

CIG Integration Policies

We present an initial set of CIG integration policies to safely and efficiently integrate comorbid CIG at execution-time. We formalize their high-level semantics in terms of the CIG Integration Ontology (CIG-IntO) and First-Order Logic (FOL) rules. These integration semantics are informed by, and operate on, the multi-level state machine introduced previously.

EquivTasksPolicy

A practitioner creates an *EquivTasksPolicy* instance to cope with equivalent tasks: e.g., both OA and diabetes guidelines recommend NSAID, but this type of drug should only be prescribed once, since it increases risk of bleeding. To that end, the practitioner identifies the integration points and instantiates an *EquivTasksPolicy* as follows (using Turtle syntax[5] with *CIG-IntO* as default namespace):

```
:NSAID_equiv a :EquivTasksPolicy ;                    (2)
   :equivTask :task_OA_NSAID ;
```

[5] https://www.w3.org/TR/turtle/

```
   :equivTask :task_Diabetes_NSAID ;
```

Here, the *equivTask* relation indicates the equivalent tasks, i.e., prescribing the NSAID medications, as integration points. Below, we elaborate on how these integration points may be identified, and the concrete integration semantics.

- Finding equivalent tasks

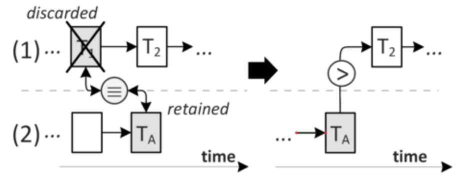
In CIG-IntO, many workflow tasks are attached to SNOMED procedures; which, if applicable, is further linked to the prescribed medication (Turtle syntax, "sno" is the SNOMED namespace). For instance, in the OA CIG:

```
:task_OA_NSAID a :WorkflowTask ;                          (3)
    :involves :presc_OA_NSAID.
:presc_OA_NSAID a sno:Prescription; sno:drugUsed sno:NSAID.
```

Since the relevant workflow tasks from both the OA and diabetes CIG are linked to a *Prescription* (subclass of *Procedure*) of the same drug (*drugUsed*), the system flags the (potential) comorbidity issue to the practitioner, indicating the two clinical tasks as integration points. Based on their medical knowledge and expertise, the practitioner may then proceed by instantiating an appropriate CIG integration policy.

- CIG integration semantics

An informed execution-time choice should be made on which equivalent task will be kept, and which one(s) will be discarded. Simply discarding an arbitrary task at *design-time* may negatively affect CIG execution, as shown in Fig. 4. Here, task T_1 was discarded at design time (Fig. 4.a) since it would have been executed last under normal circumstances. However, due to execution-time delays, task T_1 is executed first; meaning that the following tasks, such as T_2, which rely on task T_1 or its equivalent (i.e., T_A) having been executed, need to be delayed until T_A is executed (Fig. 4.b).



(a) (b)

Figure 4– Integration operations on clinical tasks.

Our execution-time approach purposefully introduces a race condition; whichever task is reached first (i.e., *activated*) during execution will be retained. In Fig. 4, this means that T_1 will be retained instead of T_A, meaning that both workflows are able to proceed without delay.

These execution-time integration semantics can be represented using the following state-transition rule:

$$EquivTasksPolicy(p), intPt(p, t_1), intPt(p, t_2), t_1 \neq t_2, \\ ActiveTask(t_1), ChosenTask(t_2), NotDiscardedTask(t_2), \\ IncompleteTask(t_2), taskState(t_2, s) \\ \rightarrow transit(t_2, 'discarded'), transitFrom(t_2, s), transitCause(t_2, t_1) \quad (4)$$

Once any integration point (*intPt*) is activated (t_1), any other integration point (t_2) part of a *chosen* branch, i.e., in line for execution, and not yet *discarded* or *completed*, will be moved to the *discarded* state (*transit* operation). The origin state (*transitFrom*) and cause (*transitCause*) are recorded as well. To simplify the formulation of integration semantics, we define complex OWL subclasses of *CIGTask*; e.g., *InactiveTask* class includes tasks with value *inactive* for property *state*.

ReplaceTasksPolicy

A *ReplaceTasksPolicy* instance is created to replace one or more tasks by safer and/or more cost-effective task(s). For instance, the thiazide diuretics prescribed for HTN may worsen glycemic control in case of HTN-Diabetes comorbidity. In clinical practice, solutions for such interactions are often nuanced, and depend on the patient's dynamic health profile. For normal cases, the dosage of thiazide diuretics should be reduced; but when creatinine clearance is low and volume control is needed, loop diuretics should be prescribed instead. This can be formulated as follows:

```
:thiazide_diuretics_diabetes a :ReplacePolicy ;              (5)
   :toReplace :thiazide_diuretics
   :replacement [
     :condition cond:default ;
     :element :low_dose_thiazide_diuretics ] ;
   :replacement [
     :condition [
       cond:operand :low_creatinine_clearance ;
       cond:conn cond:and ;
       cond:operand :volume_control_needed
     ] ;
     :element :loop_diuretics ] .
```

The *toReplace* relation indicates the integration point(s), i.e., the clinical task(s) to be replaced. The *replacement* relation indicates the conditional substitutions for the CIG task, marking (a) the condition (*condition* relation), which refers to the patient profile; and (b) the replacement task (*element* relation). The default substitution, i.e., with "default" condition, prescribes *low dose thiazide diuretics*; the second substitution prescribes *loop diuretics* in case *volume control is needed* and *low creatinine clearance* is observed.

Using our CIG framework, the practitioner can leverage LOD to identify such integration points. To that end, the *:thiazide_diuretics* task is linked to the related medication (e.g., see (3)). Based on this knowledge, there are multiple ways for the CIG framework find potential interactions:

- *Finding drug-drug interactions (DDI)*

Searching the DINTO and Bio2RDF DrugBank ontologies, the system discovers multiple DDI between Methyclothiazide and different types of insulin—if the patient was prescribed insulin for managing glucose levels, the system will flag these DDI to the practitioner together with the integration points. E.g., DrugBank encodes the DDI in question as follows:

```
drb:DB00232 dcterms:title "Methyclothiazide"@en ;            (6)
    drb:ddi-interactor-in db:DB00030_DB00232 .
drb:DB00030_DB00232 a drb:Drug-Drug-Interaction .
```

- Finding drug-illness interactions (DII)

The system finds that Methyclothiazide is a type of *Thiazide Diuretic* (SNOMED CT), which are categorized as *Hyperglycemia-associated Agents* (DrugBank); whereas Diabetes is a type of *Disorder of Glucose Metabolism* (SNOMED CT). Based on a DII knowledge source, which encodes adverse interactions between agents and disorders of that type, the system flags the DII and integration points to the practitioner.

- CIG integration semantics

When *multiple tasks* need to be replaced, as before, a choice needs to be made on which task to discard (see Fig. 4). In that case, a transition rule similar to (4) will be applied—which additionally ensures that other task(s) (i.e., t_2) are only discarded when a replacement was actually applied (see (7)). When only a *single task* is to be replaced (as in (5)), the *toReplace* task is simply discarded when it becomes active.

Based on the patient's current health profile, a substitution will be selected at execution-time, after which the *toReplace* task will be replaced in the CIG workflow (*NotApplied* ensures that a replacement is applied only once):

$$ReplaceTasksPolicy(p), intPt(p, t_1), ActiveTask(t_1), \\ replacement(p, r), condition(r, c), Satisfied(c), \\ NotApplied(r), element(r, t_2) \\ \rightarrow replace(t_1, t_2), applied(r, 'true') \quad (7)$$

SimulTasksPolicy

To simultaneously execute certain clinical tasks, the practitioner creates an instance of *SimulTasksPolicy*. For instance, many comorbid illnesses involve imaging scans, ECG and other tests (e.g., spirometry) at a point-of-care or healthcare facility; by grouping such medical tests together in time, we may improve the patient's quality of life and reduce healthcare costs. This can be encoded using *SimulTasksPolicy* as follows:

```
:simult_siteofcare_tests a :SimultTasksPolicy ;            (8)
  :simulTask [
    :involves [
      a sno:Evaluation_Procedure ;
      sno:placeOfTesting [ a sno:SiteOfCare ]
] ] .
```

In the above case, the policy aims to group in time any clinical task that *involves* any procedure of type of *Evaluation Procedure*, and with any *placeOfTesting* of type *SiteOfCare*. In SNOMED CT, medical tests such as ECG, MRI scans, spirometry are all subtypes of *Evaluation Procedure*; whereas *SiteOfCare* has subtype *Hospital*, among others. Based on the policy definition, the CIG integration framework identifies concrete integration points for the *SimulTasksPolicy*. Below, we exemplify such an integration point:

```
:task_AF_ECG a :WorkflowTask ;                              (9)
    :involves :proc_ECG_monitor .
:proc_ECG_monitor a sno:ECG_Monitoring ;
   sno:placeOfTesting sno:Hospital .
```

- *CIG integration semantics*

To group the identified integration points at execution-time, a *SimulTasksPolicy* will delay any activated *simulTask* until the other *simulTasks* have "caught up" (i.e., have also been activated)—at least, as long as it is clinically safe to do so.

In particular, the *max. (overall) delay* of the active task shows the max. period *the task may still be delayed* while safeguarding clinical safety. Hence, as long as the *max. delay* exceeds the (estimated) *activation time* of an inactive task, it will make sense, and not violate clinical safety, to delay the active task:

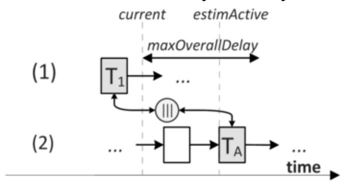


Figure 5– Example workflows for SimulTasksPolicy.

Here, the *max. overall delay* for T_1 exceeds the *estimated activation time* of T_A, meaning that T_1 can be safely delayed until T_A catches up. The following should hold: *current* + *maxOverallDelay*$_1 \geq$ *estimActive*$_A$, which is formalized as follows:

$$maxOverallDelay(t_1, d_1), estimActive(t_2, a_2), t_1 \neq t_2, current + d_1 \geq a_2 \\ \rightarrow delayCondition(t_1, t_2) \quad \textbf{(10)}$$

The *maxOverallDelay* property is calculated based on (a) the initial max. period that workflow tasks can be delayed while ensuring patient safety; and (b) any already incurred delays during execution, as well as the impact of discarded and replaced tasks. The *estimActive* property is calculated recursively from the prior task's *estimActive* time and duration.

The integration semantics are represented as follows:

$$SimulTasksPolicy(p), intPt(p, t_1), intPt(p, t_2), t_1 \neq t_2, \quad \textbf{(11)} \\ ActiveTask(t_1), delayCondition(t_1, t_2), taskState(t_1, s), \\ PendingOrChosenTask(t_2), InactiveTask(t_2) \\ \rightarrow transit(t_1, 'delayed'), transitFrom(t_1, s), transitCause(t_1, t_2)$$

In case the *delayCondition* holds for an active task (t_1) and at least one other integration point (t_2), which is part of a branch with a pending decision or a chosen branch (*PendingOrChosenTask*), the active task will be delayed.

As the CIG execution progresses and more information becomes available, integration decisions may need to be adapted or reverted. When the delay condition no longer applies, the delay operation should be reverted:

$$SimulTasksPolicy(p), intPt(p, t_1), DelayedTask(t_1), \quad \textbf{(12)} \\ transitCause(t_1, t_2), \neg(delayCondition(t_1, t_2)) \\ \rightarrow revert(t_1, 'delayed'), revertCause(t_1, t_2)$$

In case the *delayCondition* no longer holds for a *delayed* task and the reason for the delay (*transitCause*), the delay operation will be reverted. Note that, at execution time, the delay condition could become valid again at any point; in that case, the *delayed* state transition (10) would be triggered again.

Secondly, when the operation's *transitCause*, i.e., reason for the delay, eventually becomes part of a non-chosen branch (*NotChosenTask*), it becomes pointless to delay the clinical task any further, and the delay operation will be reverted:

$$SimulTasksPolicy(p), intPt(p, t_1), DelayedTask(t_1), \quad \textbf{(13)} \\ transitCause(t_1, t_2), NotChosenTask(t_2) \\ \rightarrow revert(t_1, 'delayed'), revertCause(t_1, t_2)$$

Once the previously inactive task has caught up (i.e., is *activated*), task t_1 no longer needs to be delayed:

$$SimulTasksPolicy(p), intPt(p, t_1), DelayedTask(t_1), \quad \textbf{(14)} \\ transitCause(t_1, t_2), ActiveTask(t_2) \\ \rightarrow revert(t_1, 'delayed'), revertCause(t_1, t_2)$$

When multiple medical tests need to catch up, an active medical test will be delayed multiple times. By keeping the causes of task operations (*transit*/*revertCause* properties; Fig. 2), the integration algorithm reverts an operation *only once all its causes are reverted*; as discussed in the next section.

CIG Integration Algorithm

In this section, we discuss how the CIG integration algorithm applies the CIG task operations inferred by the integration semantics. We assume an underlying reasoner that is loaded with the rules representing the integration semantics. The CIG integration algorithm is called whenever dataset changes may lead the reasoner to fire one or more rules:

- Any tasks' state changes caused by the CIG execution algorithm (see *Workflow-related task lifecycle*).
- External events indicating execution-time delays (e.g., from scheduling software), which affects the estimated times (e.g., *maxOverallDelay, estimActive*).

The *performIntegration* function iteratively applies task operations and reverts until no more operations are inferred:

```
function performIntegration()                               (15)
  do
    op ← performTaskOperations()
    op ← op ∪ performTaskReverts()
  while (op ≠ ∅)
```

The *performTaskOperations* function calls the *performTransitOperations* and *performReplaceOperations function* (not shown). We show the former function below:

```
function performTransitOperations()                         (16)
  for each inferred transit operation:
    retract current state of task (= origin state)
    assert target state for task
    assert origin state(transitFrom),cause(transitCause)
end function
```

For each inferred *transit* operation, the function retracts the task's current state, and asserts its new state, keeping its origin state and transit cause. Below, we show *performTaskReverts*:

```
function performTaskReverts()                               (17)
  for each inferred revert operation:
    retract reverted cause (revertedCause)
    if all task transit causes ≠ ∅:
      retract current task state
      assert original target state (transitFrom)
end function
```

For each inferred *revert* operation, the "reverted" cause is retracted (e.g., see (11)-(13)). When no more *transit causes* are left, the task's current state is retracted and its original state is asserted. We note that, by only reverting a *transit* operation once all its causes are reverted, we support scenarios where e.g., a medical test is delayed multiple times (see (10)).

The *CIG Client* (not detailed here) features a UI that visualizes the current state of comorbid CIG as a workflow, and allows clinicians to make decisions on task execution and integration.

Results

When deploying execution-time, comorbid CIG integration at a point-of-care, there is an expectation of timely recommendations. Hence, we evaluate our approach by measuring the performance of the CIG integration algorithm. To that end, we (a) modeled the comorbid CIG; (b) instantiated suitable CIG integration policies; and (c) executed the CIG integration algorithm to perform execution-time integration for a number of comorbidity scenarios.

We executed each experiment 10 times on a PC equipped with 8GB of RAM and an Intel® Core™ i7-3520 CPU. Table I shows the average performance results. Loading the comorbid CIG and integration policies took avg. ca. 270ms; calculating dynamic properties (e.g., *estimActive*) took avg. ca. 180ms.

Table 1 – CIG Integration Performance Results

Comorbid CIG Integration Scenario	**performance (s)**
EquivTasksPolicy	
(OA-diabetes) discard second NSAID	0.79
ConditionalReplacePolicy	
(HTN-Diabetes) in regular cases, reduce dosage; when creatinine clearance is low and volume control is needed, replace with loop diuretics.	0.94
SimulTasksPolicy	
group tests concurrently at health facility	0.9

Discussion

Table 1 shows that integrating comorbid CIG takes between 0.7 – 0.9s. We note that, since performance is mostly determined by the underlying reasoner (in our case, Apache Jena), selecting a more performant reasoner would improve performance. Nevertheless, we already consider these acceptable performance times for a consumer-grade PC.

To the best of our knowledge, the work by Anselma, Piovesan and Terenziani [4, 10] presents the first approach to focus on the temporal dimension for comorbid CIG integration. Analysis facilities allow a practitioner to analyse CIG for adverse interactions and identify solutions. To emulate our execution-time approach, the practitioner would need to frequently utilize these facilities *for any set of relevant actions*, and manually apply, adapt and revert integrations when needed. Wilk et al. [7] support a form of execution-time integration by repeating the integration process whenever new patient data becomes available. But the authors do not consider reverting integrations, and only supply two temporal revision operators. Zamborlini et al. [11] propose a rule-based approach that identifies repeated, alternative, and contradictory actions, and leverages external sources. In comparison, our work currently includes limited support for identifying adverse interactions by leveraging LOD.

Another aspect of our approach, not elaborated here due to space limitations, involves resolving conflicts between integration policies themselves. In future work, we aim to study more types of integration policies, and look into a global optimization scheme—currently, integration policies focus solely on their integration points, and disregard the global effect of their actions; e.g., a *SimulTasksPolicy* could delay a task that jeopardizes the success of more "important" policies.

Conclusions

We presented an execution-time approach and framework for comorbid CIG integration, which dynamically (a) applies CIG integrations, based on a priori defined integration policies and the latest execution-time data; (b) refines or reverts previous integrations, as more information becomes available at runtime. To provide solutions in line with clinical practice, alternative CIG integration decisions can be defined, which are conditional on the patient's up-to-date health profile. By leveraging knowledge from LOD, our framework helps practitioners in identifying adverse comorbid interactions, and formulating suitable integration policies.

References

[1] J. E. Brush, M. J. Radford, and H. M. Krumholz, "Integrating Clinical Practice Guidelines Into the Routine of Everyday Practice," *Crit. Pathways Cardiol. A J. Evidence-Based Med.*, vol. 4, no. 3, pp. 161–167, 2005. (3)

[2] National Institute for Health and Care Excellence (NICE), "Chronic kidney disease in adults," 2014.

[3] S. R. Abidi, "A Knowledge Management Framework to Develop, Model, Align and Operationalize Clinical Pathways to Provide Decision Support for Comorbid Diseases," Dalhousie University, Halifax, NS, 2010.

[4] L. Anselma, L. Piovesan, and P. Terenziani, "Temporal detection and analysis of guideline interactions," *Artif. Intell. Med.*, vol. 76, pp. 40–62, 2017.

[5] F. Real and D. Riaño, "Automatic combination of formal intervention plans using SDA* representation model," in *2007 conference on Knowledge management for health care procedures*, 2008, vol. Amsterdam, pp. 75–86.

[6] D. Riaño and A. Collado, "Model-Based Combination of Treatments for the Management of Chronic Comorbid Patients," in *Artificial Intelligence in Medicine SE - 2*, vol. 7885, N. Peek, R. Marín Morales, and M. Peleg, Eds. Springer Berlin Heidelberg, 2013, pp. 11–16.

[7] S. Wilk, M. Michalowski, W. Michalowski, D. Rosu, M. Carrier, and M. Kezadri-Hamiaz, "Comprehensive mitigation framework for concurrent application of multiple clinical practice guidelines,"*J.Biomed.Inform.*,vol.66,pp. 52–71, 2017.

[8] B. Jafarpour, S. S. R. Abidi, and S. R. Abidi, "Exploiting Semantic Web Technologies to Develop OWL-Based Clinical Practice Guideline Execution Engines," *IEEE J. Biomed. Heal. Informatics*, vol. 20, no. 1, pp. 388–398, 2016.

[9] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "OWL 2 Web Ontology Language Primer (Second Edition)," 2012.

[10] L. Piovesan and P. Terenziani, "A Mixed-Initiative Approach to the Conciliation of Clinical Guidelines for Comorbid Patients," in *Knowledge Representation for Health Care*, 2015, pp. 95–108.

[11] V. Zamborlini, R. Hoekstra, M. da Silveira, C. Pruski, A. ten Teije, and F. van Harmelen, "Generalizing the Detection of Internal and External Interactions in Clinical Guidelines," in *9th International Joint Conference on Biomedical Engineering Systems and Technologies*, 2016, pp. 105–116.

Address for correspondence

William Van Woensel, Faculty of Computer Science, 6050 University Avenue, Dalhousie University, Halifax, Nova Scotia, Canada, B3H 1W5; E-mail: william.van.woensel@dal.ca.