PCtk: A ToolKit for Managing Business Process Compliance

Aditya Ghose and George Koliadis
Decision Systems Laboratory
School of Computer Science and Software Engineering
University of Wollongong, NSW 2522 Australia
{aditya, gk56}@uow.edu.au

Abstract. Determining whether business processes comply with regulatory or legislative requirements, and understanding how best to modify processes when they are found to be non-compliant are being increasingly recognized as difficult, yet important problems. This paper proposes practical solutions to these problems by considering business processes described in the industry standard BPMN notation, and by encoding compliance requirements as business rules. We define a simple technique for analyst-mediated semantic annotation of BPMN models with effects. Design-time compliance checking then reduces to simple traversal of these effect-annotated process models. More importantly, we are able to identify minimally different process models that restore compliance, or minimal sources of inconsistency, which, if appropriately modified, would lead to compliant processes.

1 PCTk Example: Evaluating “Screen Package” Process Resolutions

We will use Figure 1: a simple BPMN “Screen Package” process owned by a Courier Organization as a motivating example, with a functional requirement stating that: $(CR_1)$ “Packages Known to be Held by a Regulatory Authority must not be Routed by a Sort Officer until the Package is Known to be Cleared by the Regulatory Authority”.

Figure 1 takes on the following immediate effect annotations: “Scan Package” - Performs(SortOfficer, Scan, Package); “Assess Package” - Performs(SortOfficer, Assess, Package)$\land$Knows(RegulatoryAgent, Package, Status, Held); “Route Package” - Performs(SortOfficer, Route, Package)$\land$Knows(SortOfficer, Package, Location, Forwarding); “Handle Package” - Performs(SortOfficer, Handle, Package)$\land$Knows(RegulatoryAgent, Package, Status, Clear); “Update Status” - Performs(SortOfficer, Update, PackageStatus); and, a General Rule $(GR_1)$ - $\forall a : Actor$ Knows$(a, PackageStatus, Held) \iff \neg$Knows$(a, Package, Status, Cleared)$.

In our semantically annotated example (Figure 1 and Table ??) we can simply determine that the “Route Package” node will induce an effect scenario where
both \( \text{Knows(RegulatoryAgent, Package, Status, Held)} \land \text{Performs(SortOfficer, Route, Package)} \) is true upon accumulation. It is also easy to see that our aforementioned compliance rule \( CR_1 \) is violated.

Figures 2 (\( R_1 \)) and 3 (\( R_2 \)) describe two simple resolutions of the inconsistent “Screen Package” process in Figure 1 (\( O \)). First of all, \( R_1 \) and \( R_2 \) trivially share all their nodes with \( O \), and therefore, no comparison can be made across this structural dimension. Next, we determine a significant edge difference between \( R_1 \) and \( O \), including the “Handle Package” \( \rightarrow \) “Route Package” edge. \( R_2 \) also differs with \( O \) across some edges including “Update Status” \( \rightarrow \) “Route Package”.

Since no inclusion relationship exists, we cannot differentiate \( R_1 \) and \( R_2 \) w.r.t. structural proximity to \( O \).

Next, we determine their semantic difference based on their final and intermediate cumulative effects. We can determine that the final cumulative effect of both \( R_1 \) and \( R_2 \) result in two effect scenarios such that \( R_1 \) actually remains identical to \( O \). \( R_2 \) on the other hand receives the additional effect of \( \text{Performs(SortOfficer, Route, Package)} \) on the effect scenario now generated by placing the “Route Package” activity in line with both process trajectories.

Next, we turn to the intermediate cumulative effects. We can establish that \( R_1 \) actually differs across only the scenario at “Handle Package” with the effect
Fig. 3. Resolved Package Screening Process \((R_2)\)

*Performs* (*SortOfficer, Route, Package*), whereas \(R_2\) actually differs w.r.t. the scenarios at: “Handle Package”, with *min* difference of 1, and *max* difference of 4; “Update Status”, with *min* difference of 1, and *max* difference of 5; “Route Package”, with *min* difference of 1, and *max* difference of 5.

We may either take the fact that there is a 1-to-3 difference between the impact of change between \(R_1\) and \(R_2\) as indication to choose \(R_1\), or in cases where a more finer grained understanding is needed, we can analyze the *min* and *max* differences in more detail.

To illustrate the propagation of quality of service measures we use the following scales: *reliability* \((R)\), measured as the number of successful completions per set of requests (i.e. \([0, 1], \max, \cdot, 0, 1\) assuming independence); *security* \((S)\), measured as the length of the encryption scheme (i.e. \([N^+, \max, \min, 0, +\infty]\)); and *cycle time* \((CT)\) measured as the average number of seconds to completion (i.e. \([R^+, \min, +, +\infty, 0]\)). Each task in our examples are annotated with the following values: “Scan Package” - \(R = 0.98, S = 128, CT = 20\); “Assess Package” - \(R = 0.96, S = 128, CT = 30\); “Route Package” - \(R = 0.94, S = 64, CT = 600\); “Handle Package” - \(R = 0.88, S = 128, CT = 50\); “Update Status” - \(R = 0.98, S = 128, CT = 10\).

As discussed, we determine the cumulative quality of service for an effect scenario by working through the path history for that scenario. Let the path histories for our examples \((R_1\) and \(R_2\)) be: \(R_1\): 1 : \((SP, AP, HP, RP, US)\), 2 : \((SP, US)\); \(R_2\): 1 : \((SP, AP, HP, US, RP)\), 2 : \((SP, US, RP)\).

We accumulate the measures in using the combination operator of each scale, leading us to the following cumulative evaluations for our examples: \(R_1\): 1 : \(R = 0.76, S = 64, CT = 710\), 2 : \(R = 0.96, S = 128, CT = 30\); \(R_2\): 1 : \(R = 0.76, S = 64, CT = 710\), 2 : \(R = 0.90, S = 64, CT = 630\). Therefore, we determine that \(R_1\) performs the same or better than \(R_2\).