Towards Formalizing Resource Based Non-Conformance in Business Processes

Sean Thompson
La Trobe University
sean@nostin.com

Torab Torabi
La Trobe University
T.Torabi@latrobe.edu.au

Abstract

The research presented in the literature thus far on process deviations, inconsistencies and general non-conformance have all been very generic, process wide methodologies. Their application has also been limited to one domain, such as the software, business or manufacturing process. In this paper we differentiate between activity/general process non-conformance and resource based non-conformance which we treat as different and the mechanism for testing is treated in a different way. This research is aimed at identifying instances of non-conformance between an instantiation of a generic process and its associated process model by examining only the resources specified and observed. We propose a conceptual design model illustrated with a simple case study on this topic which although is not based on a software engineering process, illustrate simply how our model can benefit a real world situation across many domains alongside software engineering.

1. Introduction

A lot of research has been conducted in the detecting of non-conformance between process prescriptions and their instantiated enactments. What has not been addressed in the literature (to our knowledge) is an examination of non-conformance in reference to the resources used by processes and the expected resource usage prescribed in the process model. There are two immediate benefits in a framework detecting non-conformance based on resources such as the one we are presenting in this paper. Initially, examining the tangible assets relating to a process both before and after its enactment is usually an excellent way of judging the value and/or effectiveness of the process. In most cases, we can look at what we have on hand before we begin a process and after it has concluded, hope that we are in a better position than where we started. Inspection of any assets which the process has affected is usually a good way of easily discerning this – regardless of the process domain. On the second part, it is much easier to detect non-conformance in something tangible like process resources than something abstract like actions in process activities. With this in mind, we present in this paper a simple framework in which the resources *prescribed* can be compared with the resources actually *used* to detect non-conformance between a process specification and its enactment.

There has been research conducted in non-conformance or "deviation" detection in processes, mainly onward from the early 1980s [2, 3]. None of them however, noted an inherent difference as we do in this paper between the activity domain and the resource domain in the process. This includes our own research which has been presented both in [4] and [5]. The approaches seen in the literature so far range from *process discovery* style approaches to fuzzy logic style approaches. Process discovery methodologies such as the frameworks described in [7] and [8] are based upon discovering the process model by examining actual return data from enacted processes. Once the model is discovered, it may be easily compared with further enactments to detect discrepancies because the discovered model should be on the same level as the enacted/compared model is observed. Methodologies such as the approach described in [10] are based on fuzzy logic and fuzzy sets theory also compares a reference model (which is purported to be a "flawless" execution) to enactments to detect discrepancies. The authors note that such a "flawless" execution of the process seems unlikely or perhaps even impossible which almost guarantees non-conformance regardless of how the process is enacted.

Generically speaking, a process is a "set of logically related tasks formed to achieve a defined …

outcome" as Davenport defines in [17]. According to [18] and [19] a process may also include machines, methods, rules, organizational structures, sub-procedures and computerized tools to aid in achieving its goal(s). Whilst accepting this assertion, for the purposes of this research, we define a process as a set of activities which are the individual tasks of which the process is comprised. Also included are "the resources used by activities and the human or automated actors who perform these activities" [20]. These activities may be enacted simultaneously, overlapping or in parallel [7], [11] and are assigned to *actors* who are charged with the responsibility of enacting them (as in [12]). Also, to aid in keeping the framework clean and precise, we have kept *resources* as independent as possible and connected them logically under the process activities, which use, consume and generate them.

When we refer to *non-conformance* in this paper, we are referring to an instance where an enactment of such a process as defined above does not conform to its specification. According to Bahrami [6] the specification for a process should include "the rules, constraints, attributes, and relationships of the activities, participants, roles, and informational items (such as documents) instrumental to the workflow". It should also include a definition on how the process should behave, where the process should be performed and when. So a good prescription for a process should not only define the scope of the behavior the process should follow but also its environment.

There are two types of non-conformance, being *deviations* and *inconsistencies* which are considered separately. The two concepts are differentiated in [1], where deviations relate to transition between states or activities and inconsistencies relate to the values inherent to states. Therefore, "deviations" are not relevant to resource non-conformance and hence also to this body of research as we do not address activity based non-conformance in this paper. The authors in [11] state that according to their eServices in Business Processes model, activities require "resources" for their enactment. They also imply that humans can be considered a resource; however they do not define what they actually mean by the term "resource". In this body of research we have conducted, when we refer to a "resource" we are referring to any tangible asset that the process consumes or generates to achieve its purpose. If "activities" are the actions within a process, then "resources" are what these actions are applied to, in the scope of this framework.

Although there has been quite a significant amount of research conducted in the process resources realm, none of the frameworks presented thus far has aimed at detecting non-conformance. The research in relation to process resources in the literature so far tend to keep to topics like process simulation to ascertain the most efficient allocation of resources such as [13] which was tested in a hospital emergency room or the flexible allocation of limited resources over a variety of dynamic demands for them as in [14]. Research has been presented on evaluating resource value with respect to the process or processes they are associated with as presented by Roy et al [16] in their paper on IT outsourcing. Experiments have even been presented like the research presented in [9] where required resources for a particular process are deliberately limited so as to force people to find a more efficient method of achieving the process goal, which can lead to long term process improvement.

In the coming sections of this paper, we will first define a resource in the context of this framework and domain of research in section 2. We will explain what we mean when we refer to a "resource" and how we define it conceptually and how we store it in the model. In section 3 we illustrate the key issues and challenges we must resolve in order to successfully detect resource based non-conformance and ultimately add value to the process we apply this model to. We explain how the resource definition data needs to be structured and stored in relation to the rest of the process prescription in order to address these issues. Section 4 deals with the actual flow of rules which compare actual data to the prescription to detect non-conformance. These rules are defined and a flowchart of how they should be implemented is given. We then present a small indicative case study of how such a framework can be used in real world situations in section 5. Section 6 concludes the paper and provides an indication on the way forward with our future research.

2. Defining a Process Resource

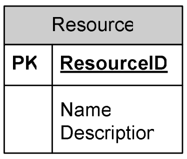
Considering the large amount of work presented in the literature on resources in general, it is imperative we define exactly what we are talking about when we refer to a "resource". Hu et al [15] suggest that a resource could be a human, machine or application. Incidentally, they also consider resources as an entity which can perform tasks or activities instead of the other way around as we (and as most others) define it. In the context of this framework, we consider a resource to be something *tangible* that a process may use during its execution. This enables us to keep a clean yet comprehensive and concise methodology when examining a given processes resources.

Intangible resources are attributes associated with the process that cannot be quantitatively observed like tangible resources can. We have identified the following process properties that in some models such as [11] or [15] may be considered a resource:

- Time Duration: the time constraints we may place upon a process or its associated activities. The minimum and/or maximum time duration they can go on for.
- Time Relative: There may be constraints as to when a process, its activities or resources may be invoked. For example, perhaps the pizza shop is closed after 11pm and so a pizza making process enactment would be inconsistent with its model it were to be performed after this time.
- Humans: In our framework, humans are characterized as *actors* to whom the responsibility of the enactment of a process activity resides.
- Human Skills: An attribute associated with the actor and factored into his or her role in performing the relevant activity.

Although some may contend that these attributes qualify as resources and should be included in the model, we have actually considered the aforementioned concepts and included them in our extended process model by relating them to the process activities. As such, they are not explained in detail in this paper.

Essentially, we have attempted to keep resources as loosely coupled from the process itself from other concepts in the model as possible. In our extended approach, we attack the problem from two separate angles – the process activity conformance and the resource conformance. The activities are obviously closely related to the process however the resources may not necessarily be, as they exist without the process and may be used by other processes also. With that said, our resource class definition is quite simple and is shown below in figure 1. The method we use to relate the resources to the process is explained in further detail in section two.



Resource	
PK	<u>ResourceID</u>
	Name Descriptior

Figure 1 – Resource Definition

3. Issues Faced

Throughout this section, we will explain the issues we have identified and addressed with respect to resource based process non-conformance detection. These issues and their resolutions will be illustrated by a simple pizza making example process. Also explained in this section, is the framework we have used to relate the resources to the process and how data is stored using this methodology. The structure flow of the rules used to make comparisons between the resource prescription and the actual data is also explained and formalized. The four subsections in this section of the paper steps through the issues one by one and updates the process prescription data structure incrementally as we address each issue. In the last subsection 3.4 we illustrate the final data structure that we will use to prescribe process resources and their relationship to the process activities which are using them.

3.1 Relating resources to the process

Consider the class diagram in figure 2 below:

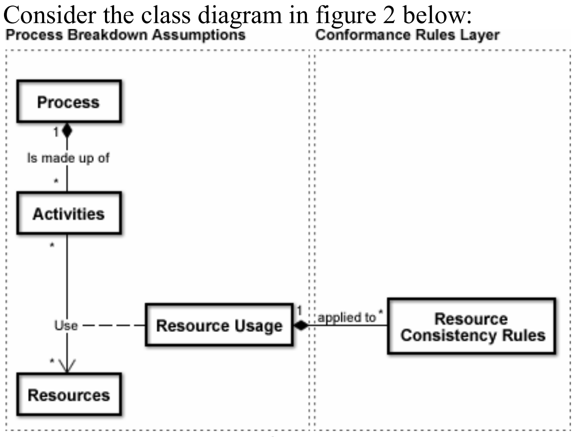


Figure 2 – Class Diagram

The diagram illustrated in figure 2 depicts the definition for how a process would prescribe the required resources to be used in its enactment. We have created an intermediary class called "Resource Usage" to specify which activities should use which resources and how they should be used. It is the prescription information in the "Resource Usage" class which will be compared against the actual enacted resource data to detect the non-conformance. This enables us to easily change how resources can be used in the process, adding new resources or removing unneeded ones without touching any actual resource definitions. We only define how an existing resource should behave in relation to an existing activity, and this behavior is constrained to the resource usage class.

To begin with, we can store the data for prescribing resource-activity usage as shown in figure 3:

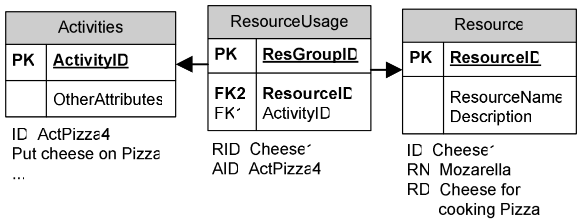


Figure 3 – Beginnings of the Resource to Activity data relationship

In figure 3, we show a simple mechanism for relating process activities to resources. Given this data structure, we can see that the records apparent in the *ResourceUsage* table will show which resources are supposed to be used by which activities. When we collect actual data on the activities used by a process then we can define a simple set of rules or queries to compare the actual data with the prescription data as shown in figure 3 to determine whether or not the activity has used the resources it was supposed to and not used resources it was not supposed to. The rules for making these comparisons are explained further in section 4.

3.2 Resource Types

As with most processes, irrespective of their domain of application, resources will be used in different ways. The first challenge we addressed in this model was conceiving a way to represent different resource types. The first attribute we allocated to resources was the "consumable" type. Using the pizza making example, if we can consider "cheese" a resource inherent to creating that pizza, then this resource is consumable, because after the pizza is made we will be left with less available cheese than which we started. Furthermore, we can also say that this resource has been *consumed* as opposed to *generated* because we are left with less than we started with. Conversely, if we consider an activity such as "buy cheese" in some such process, then the resource would have been generated because after the completion of the activity, the amount of the resource available will have increased.

Now what about resources which are not consumable? In the same example process, there may be an activity of the description "cook pizza". Suppose in this activity, a required resource may be an oven. When the activity is over, the same oven will still be available to this and other processes so it has not really been *consumed* as much as used. For these types of resources, we consider to be *non-consumable*.

These extra attributes can be added to our *ResourceUsage* class as depicted previously in figure 3. Subsequently, as shown in figure 4 below, the *consumable* attribute has been added to our definition. The "Consumable" attribute added to the class can only have three values: *Non-Consumable*, *Consume* and *Generate*. These are depicted in the grey box in figure 4.

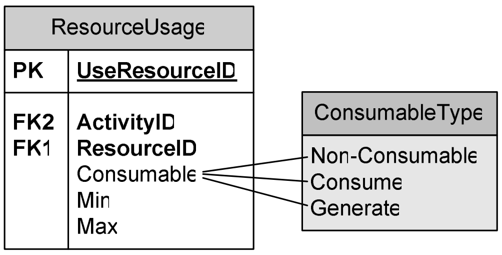


Figure 4 – Resource Consumption

In this regard, if a resource is considered to be *consumable* i.e. it can generate or consume the resources it uses, then a minimum and maximum boundary value can be applied to constrain how many of the resource a specific activity may consume or generate. This of course is redundant if the resource is of type *non-consumable*.

3.3 Exception Typed Resources

Let's say, for example, an activity with the description "sprinkle cheese on the pizza" was apparent within a process. This activity may use the resource "Mozzarella" as a *standard* resource but if Mozzarella was not available we could use Cheddar. It could be argued that the quality of the pizza may not be compromised if an alternate resource was used, however if it is considered acceptable but not ideal we classify the resource type as exceptive. The reason for this is the effect cumulative exceptions may have on the quality of the outcome of the process. For example, the same could be said if we used self-raising flour instead of flour in the making of the pizza, or red peppers instead of green peppers or tomato sauce instead of tomato paste. Singularly, these changes may not make too much of a difference, however all together collectively these less-than-ideal resource usages could ruin the quality of the end product.

Therefore, we can include another attribute in our resource usage class which defines the resource as being either *standard* or *exception* typed. Of course, some resources will be standard for some processes and activities and exceptive for others, so the UsageType is defined in the Resource Usage class, specifying that a given resource will have a UsageType applicable only for the relevant activity. The Usage Type only relates to the resources usage, not to the resource itself.

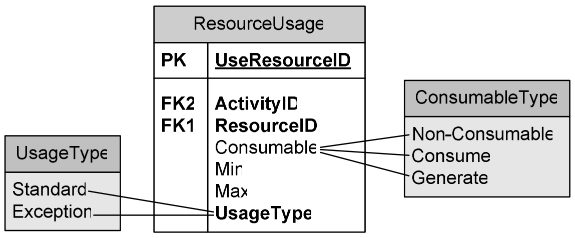


Figure 5 – Usage Types

3.4 Shared and Conflicting Resources

The final issue facing the structure of this framework is how to handle shared and conflicting resources. Let's use the "sprinkle cheese on the pizza" process activity as an example. Suppose this activity required a resource of "cheese" and there were as a choice "Mozzarella" as a standard usage type and "Cheddar" as an exception type available. We need to implement a mechanism for handling a) if both were used i.e. if Mozzarella was sprinled on (which would be legal under the process prescription) and then Cheddar was used. These resources are in conflict because the pizza only required one cheese to be sprinkled yet two were used where, if used separately would not have been a problem but both used at the same time yields too much of the resource. The model needs to recognize and allow for this. Also b) if the correct amount of "cheese" was used, but it was a mixture of both resources i.e. some Mozzarella was used and some Cheddar but the sum of which was an adequate amount – this is a shared resource.

We have handled this by introducing another intermediary entity that groups the resource usage records. This way, for each activity a resource group can be specified for every resource the activity requires. Applicable resources in the resource usage class can then be classified into the group. This means, however that the attributes added previously in sections 3.2 and 3.3 have had to be restructured. For this new structure to work, the *Consumable* type along with its associated *min* and *max* attributes are now inherent to the *ResourceUsageGroup* entity. This is so we can specify for the activity which resource groups is required and whether the resource is consumable or not along with the minimum and maximum usage boundaries. When the actual resource is related into the resource group for the activity, the usage type is unique for that specific resource in each separate group, so *usage type* is now inherent to the ResourceUsageType entity.

This methodology is illustrated with an example in figure 6 below:

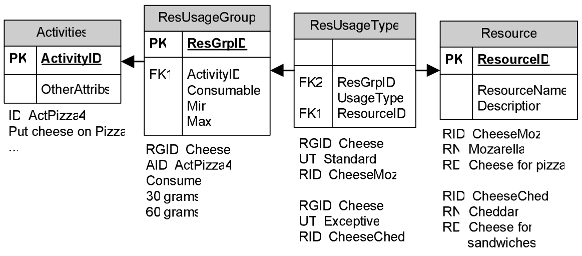


Figure 6 – Resource Usage Groups

Figure 6 shows the complete data structure for storing resource usage prescriptions in a process. It also has some dummy data underneath each entity to illustrate what records may be stored in these tables and how they would relate to each other in an actual situation.

4. Comparison Engine

This section is devoted to the sequence of rules we have implemented to test the predefined process prescription data structure. These rules have the purpose of querying the prescription data against actual process data and detecting instances of non conformance which is then stored. The "actual" process data which is recorded and compared with we call the *observed process* in concordance with [22]. The rule sequence is presented in two flows which are generic and applicable to any process which facilitates our framework. These rule sequence flows define and illustrate how the rule queries are to be implemented in order to detect and store instances of non-conformance between the process prescription and its enactments.

Considering the issues explained in section 3 and the subsequent framework devised which is based on it, we now require a flow of rules which compares the prescription data to the actual data to detect instances of non-conformance between them. These rules are generic and apply to any process which adopts this framework. The rules are structured into two flows from separate standpoints. The first is a series of checks based on actual resources observed used within an activity against the prescription. The second and simpler flow then checks what was prescribed against what was observed. These flows are explained separately in sections 4.1 and 4.2.

4.1 Recording Instances of Non-Conformance

When an instance of non-conformance is identified in either of these two flows, then a record is made in a log which stores all non-conformance instances for the process. We store each record in a data table which takes the following form as shown in figure 7:

NonConformanceLog	
PK	**<u>LogID</u>**
FK1 **FK2**	**ProcessID** **ActivityID** NonConformanceType OffendingValue SeverityRating ActorResponsibleID

Figure 7 – Non Conformance Log

Although this structure is simple, it facilitates some useful information about the types of non-conformance being detected within a process. We can easily see whether certain non-conformance types are reoccurring overly much and identify potential trouble spots in the process prescription with just a cursory look at the non-conformance data. The paper presented in [21] notes that "while the importance of a repository is often acknowledged, the difficulty of producing an *effective* tool is often underestimated". This is the subject of some research we are presently conducting with this repository, in how we can take a repository of useful data such as the one illustrated in figure 7 and provide a tool to extract beneficial and meaningful information concerning a given process.

4.2 Checking the observed resource usage against the prescription

Once, assuming that we have an adequate amount of actual enacted process data, we need to compare this data against what was prescribed for the process. We begin by taking the observed data and comparing it to the prescription. The aim in this flow is to:

a) detect resources which were used that were not supposed to be;
b) if the resource was supposed to be used, and it was a consumable type, it either generated or consumed the resource as was prescribed; and
c) the amount consumed or generated was within the boundaries set by the min and max attributes (if they were set).

The diagram below in figure 8 explains the flow of rules that we follow to make these checks:

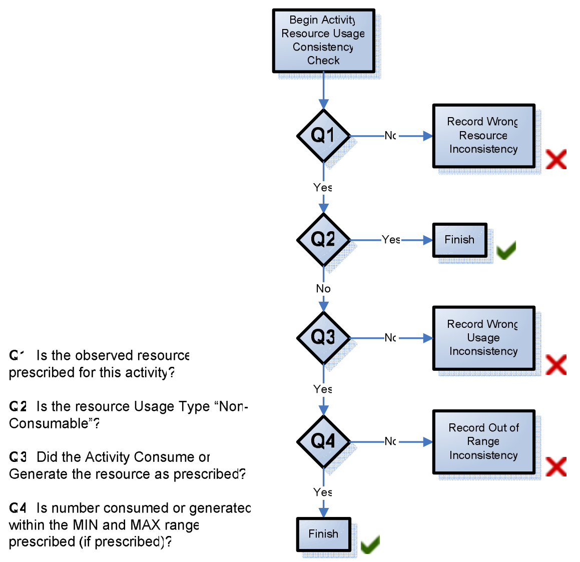

Begin Activity Resource Usage Consistency Check
Q1
No
Record Wrong Resource Inconsistency
Yes
Q2
Yes
Finish
No
Q3
No
Record Wrong Usage Inconsistency
Yes
Q4
No
Record Out of Range Inconsistency
Yes
Finish
Q1 Is the observed resource prescribed for this activity?
Q2 Is the resource Usage Type "Non-Consumable"?
Q3 Did the Activity Consume or Generate the resource as prescribed?
Q4 Is number consumed or generated within the MIN and MAX range prescribed (if prescribed)?

Figure 8 – Flow of Rules 1

In figure 8 above, Q1 is checking whether or not the observed resource was prescribed at all. If it was, and we are storing the prescriptive data in a relational database, then there should be a record in the *ResourceUsageType* table with both the activity ID of the activity we are checking and also the resource ID that was observed. If not, then the resource was not prescribed and this is an inconsistency which is logged.

Q2 in the diagram checks whether or not the resource was prescribed as being "non-consumable".

If it was, then we can safely stop the flow here because the rest of the checks in the flow (Q3 and Q4) relate to resource prescriptions which are either "generate" or "consume" usage types.

Q3 checks whether the observed resource usage was "consumed" if it had a prescribed usage type of "consume" or alternatively that it was "generated" when the prescribed usage type was "generate". If the prescribed usage type and the observed usage type match, then we continue on, but if not then we record a non-conformance instance being a wrong usage type.

The final check in this flow, after everything else has been confirmed as fine and only if the prescribed resource type is either "consume" or "generate" is whether or not the amount consumed or generated is within the minimum and maximum boundary amounts prescribed (if prescribed at all).

Along this flow, we record non-conformance instances as they are detected and upon conclusion, we begin the second flow which is described in section 4.2.

4.3 Checking the prescription against the observed resource usage

The second flowchart conversely completes the non-conformance testing by checking the *prescribed* resources for the activity against what has actually occurred in an attempt to detect omitted resources. It also checks for conflicting and shared resource inconsistencies as described previously in section 3.4:

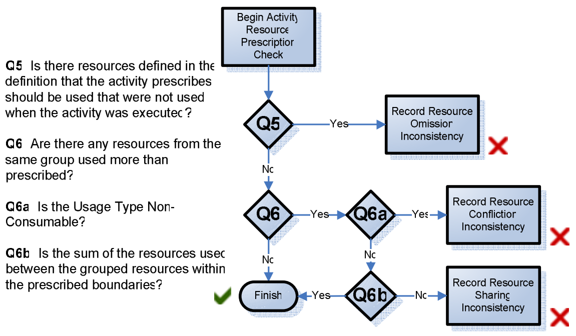

Begin Activity Resource Prescription Check
Q5 Is there resources defined in the definition that the activity prescribes should be used that were not used when the activity was executed?
Q6 Are there any resources from the same group used more than prescribed?
Q6a Is the Usage Type Non-Consumable?
Q6b Is the sum of the resources used between the grouped resources within the prescribed boundaries?
Q5
Yes
Record Resource Omission Inconsistency
No
Q6
Yes
Q6a
Yes
Record Resource Confliction Inconsistency
No
No
Finish
Yes
Q6b
No
Record Resource Sharing Inconsistency

Figure 9 – Flow of Rules 2

The first thing the second flow checks with the Q5 test is whether or not there were any resources that were prescribed that was not used. The methodology to perform this is to check every resource that was prescribed for the activity and then ensure that a matching resource record is available in the actual process data. If not, then a resource is missing and this constitutes an instance of non-conformance and is recorded.

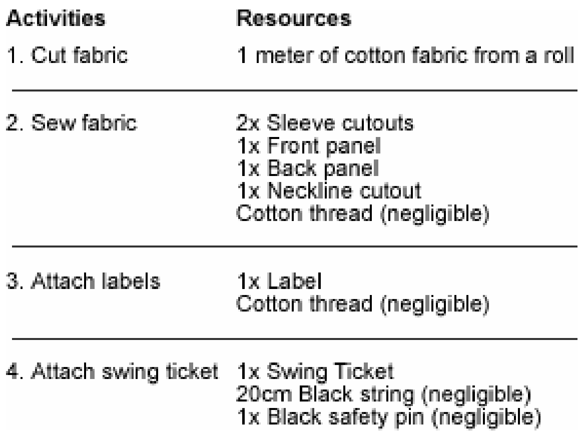
The Q6 checks in this flow relate to the conflicting and shared resources described in section 3.4. Logically, if only one resource from each prescribed resource group is used, then there can be no shared or conflicting resource inconsistencies. So, if we detect that more than one resource in a group has been used in the enactment then we need to check for this, which is what Q6a and Q6b performs.

5. Case Study

We chose a simple process with which to test our initial implementation of this structure, which is a t-shirt production process. The process involves the basic steps that the company we have chosen PHEROMONE™ (also explained in [5]) take in order to produce t-shirt garments for sale. This process is simple enough so that the methods, activities and resources are basic enough to easily and quickly understand whilst also illustrating how our framework can be implemented in a real world environment – albeit not native to the realm of software engineering.

As a basic process overview, the process has certain characteristics and environmental aspects that should be mentioned as to properly explain the scope. We begin with what constitutes 500m of 140GSM 100% cotton fabric in 5 different colours. We have been advised that based on the manufacturers experience, 100m of fabric of this type and weight (140GSM cotton) that approximately 100 t-shirts can be created in the standard men's size and using the standard 2:2:1 ratio of 40 large, 40 medium and 20 small – with only scrap amounts of fabric left over. We also have an adequate supply of garment labels and swing tickets which have been pre-purchased and provided to the manufacturer to attach to each shirt. The manufacturer has also advised that the thread used to sew each cut panel of fabric together is trivial and not charged for, so it is therefore not included in our process model as it has no impact on the process.

This process can be broken down into the following basic activities and their associated resources (if any) as depicted in figure 10:



Activities	Resources
1. Cut fabric	1 meter of cotton fabric from a roll
2. Sew fabric	2x Sleeve cutouts 1x Front panel 1x Back panel 1x Neckline cutout Cotton thread (negligible)
3. Attach labels	1x Label Cotton thread (negligible)
4. Attach swing ticket	1x Swing Ticket 20cm Black string (negligible) 1x Black safety pin (negligible)

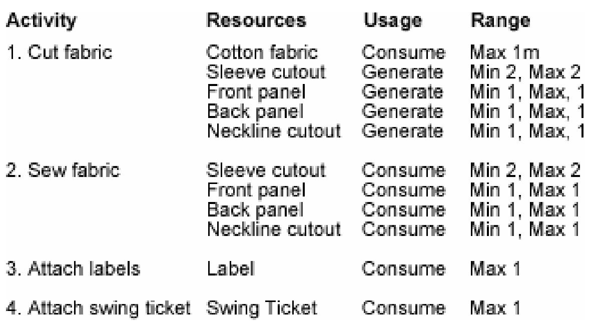
Figure 10 – Tshirt manufacturing process

Along with the process characteristics which have been explained in figure 10, there are two immediate concerns which must be noted when applying our framework. The first is the resources labeled "negligible". In processes such as the aforementioned, seemingly relevant resources such as cotton thread are in reality not. This is because of the structure of how the garments are produced for the company places the onus of the thread and sewing equipment onto the external manufacturer who bears this trivial cost in its entirety. Therefore, a degree of familiarity with the process and the applicable framework is required in order to implement a methodology such as ours successfully.

The second is that a person charged with modeling the process to a framework such as ours would need to know certain things about how the process would relate to the model which are not immediately clear. These things could include the omitted irrelevant aspects of the process such as fabric sourcing, logistics, cosmetic design and printing, washing, pressing etc… or which resource types should be classified as "negligible" and also omitted or even process specific characteristics, such as how the resource which was once 1 meter of cotton fabric in activity 1 turns into front, back, neck and sleeve panels, a separately considered resource used in activity 2.

Fortunately, the run did not experience any problems and the garments we inspected were of merchantable quality. However, one potential consequence became immediately apparent which we had admittedly foreseen anyway, which was the critical need for a complementary tool to observe and record event data from the observed process. Unfortunately, such a tool would probably need to be specific to each process we would use our framework on and thus unfeasible for us to build in a generic fashion. Despite this, we managed to record manually and reasonably the required data observed from this process.

The prescription repository was filled with the same 4 activities as seen in figure 10, however the related resource repository was slightly different as previously explained, beginning with an entry for all resources to be used throughout the process (whether presently available or not):



Activity	Resources	Usage	Range
1. Cut fabric	Cotton fabric	Consume	Max 1m
	Sleeve cutout	Generate	Min 2, Max 2
	Front panel	Generate	Min 1, Max, 1
	Back panel	Generate	Min 1, Max, 1
	Neckline cutout	Generate	Min 1, Max, 1
2. Sew fabric	Sleeve cutout	Consume	Min 2, Max 2
	Front panel	Consume	Min 1, Max 1
	Back panel	Consume	Min 1, Max 1
	Neckline cutout	Consume	Min 1, Max 1
3. Attach labels	Label	Consume	Max 1
4. Attach swing ticket	Swing Ticket	Consume	Max 1

Figure 11 – Resource Repository

With the repository set up like the following, we can easily and quickly see when resources fall out of scope. For example, if we make a t-shirt but use more than 1m of fabric to cut the panels, we can spot the non-conformance once the activity is complete for that particular shirt instead of having to wait until the entire roll of fabric has been used before discovering there is not enough to complete the job.

Conversely, another benefit of this type of model is we can see if considerably less fabric is being used (which incidentally did not happen) than the stipulated maximum. However the major benefit as we contend in this paper is on the usefulness of having a proper, well defined structure which can easily and definitively compare a prescription to an observation. Occurrences such as a resource being generated or consumed when the opposite was prescribed, or resource usage which is out of range of the prescription is, using this model quite easily detected.

6. Conclusion and Future Work

In this paper we have outlined a detailed methodology on how process resources may be observed and compared using a structured flow of rules to detect non-conformance between actual process data and its prescription. This framework is useful for structuring a process resource usage prescription in order to easily compare actual process resource data against it to detect non-conformance, regardless of how the actual data is returned and recorded. The logical structure of storing prescription data was presented and the flow of rules for comparing the prescription data with the actual data was described analyzed. We have also provided an example "pizza cooking" example in order to illustrate and better explain the model we have implemented as well as detailing a simple case study in which we tested the apparent applicability and usefulness of our methodology.

Further from this research we would like to formalize this approach in a generic fashion and also apply the same kind of methodology to process activities and draw the two approaches together for a more complete framework for detecting non-conformance between a process prescription and actual enactments of it. We would also like to test a more comprehensive and formal implementation of this framework on a variety of real world processes within different domains. We can then analyze the results to further examine how a methodology such as the research presented in this paper can be used to provide useful real life assistance to actual implemented industrial processes.

References

[1] G. Cugola, E. Di Nitto, A. Fuggetta, C. Ghezzi; "A framework for formalizing inconsistencies and deviations in human-centered systems", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 5 Issue 3, ACM Press, July 1996.

[2] V.R. Basili, F.E. McGarry, R. Pajerski, M.V. Zelkowitz; "Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory", *Proceedings of the 24th International Conference on Software Engineering*, ACM Press, May 2002.

[3] A. Fuggetta, "Software Process: A Roadmap", *Proceedings of the Conference on The Future of Software Engineering*, ACM Press, May 2000.

[4] S. Thompson, T. Torabi, P. Joshi, "A Framework to Detect Deviations During Process Enactment", *6th IEEE International Conference on Computer and Information Science*, IEEE Computer Society Press, Melbourne, Australia, July 2007.

[5] S. Thompson, T. Torabi, "A Process Improvement Approach to Improve Web Form Design and Usability", *The 3rd Ubiquitous Web Systems and Intelligence Workshop (UWSI 2007) Colocated with DEXA 2007*, Regensburg, Germany, 3-7 September 2007.

[6] A. Bahrami, "Integrated process management: from planning to work execution", *Proceedings of the IEEE EEE05 international workshop on Business services networks BSN '05*, IEEE Press, March 2005.

[7] M. Huo, H. Zhang, R. Jeffery, "An Exploratory Study of Process Enactment as Input to Software Process Improvement"**,** *International Conference on Software Engineering*, 2006.

[8] J.E. Cook, A.L. Wolf; "Discovering models of software processes from event-based data", *ACM Transactions on Software Engineering and Methodology (TOSEM)***,** Volume 7 Issue 3, July 1998.

[9] L.J. Burnell, J.W. Priest, J.R. Durrett; "Reviewed papers: Assessment of a resource limited process for multidisciplinary projects", *ACM SIGCSE Bulletin*, Volume 35 Issue 4, ACM Press, December 2003.

[10] S. Cîmpan, F. Oquendo; "Dealing with software process deviations using fuzzy logic based monitoring"**,** *ACM SIGAPP Applied Computing Review***,** Volume 8 Issue 2, ACM Press, December 2000.

[11] Y. Rezgui, F. Marir, G. Cooper, J. Yip, and P. Brandon, "A Case-Based Approach to Construction Process Activity Specification", *Intelligent Information Systems IIS* '97, 8-10 Dec. 1997, pp. 293 – 297.

[12] M. Dowson, B. Nejmeh, and W. Riddle, "Concepts for Process Definition and Support", *Proceedings of the 6th International Software Process Workshop*, IEEE Computer Society Press, Hakodate, Japan, October 28-31 1990.

[13] J. April, M. Better, F. Glover, J. Kelly, M. Laguna, "Enhancing Business Process Management With Simulation Optimization"**,** *Proceedings of the 2006 Winter Simulation Conference*, Monterey, California, USA, December 3-6, 2006.

[14] U. Deshpande, A. Gupta, A. Basu, "Adaptive Problem Solving among Business Organizations through Flexible Resource Allocation"**,** *International Conference on Advanced Computing and Communications (ADCOM)*, 20-23 Dec. 2006 Page(s):382 – 387.

[15] L. Hu; Y. Hu; "Resource perspectives in the context of process management of product development", *7th International Conference on Computer-Aided Industrial Design and Conceptual Design*, 2006. CAIDCD '06, 17-19 Nov. 2006 Page(s): 1 – 5.

[16] V. Roy, B.A. Aubert, “A resource-based analysis of IT sourcing”, *ACM SIGMIS Database*, Volume 33 Issue 2, ACM Press, June 2002.

[17] T. Davenport, “Process Innovation: Re-engineering Work through Information Technolog”y, *Harvard Business School Press*, Boston MA, 1993.

[18] A. Blyth, “Business process re-engineering: What is it?”, *ACM SIGGROUP Bulletin*, Volume 18 Issue 1, April 1997.

[19] W.A. Florac, A.D. Carleton, *Measuring the Software Process: Statistical Process Control for Process Improvement*, Addison-Wesley, 1999.

[20] A. Caetano, M. Zacarias, A.R Silva, J. Tribolet, “A Role-Based Framework for Business Process Modeling”, *Proceedings of the 38th Hawaii International Conference on System Sciences*, Hawaii, USA, 2005.

[21] K. Schneider, J.P von Hunnius, “Experience reports: process and tools: Effective experience repositories for software engineering”, *Proceedings of the 25th International Conference on Software Engineering ICSE '03*, IEEE Computer Society, May 2003.

[22] K. Mohammed, L. Redouane, C. Bernard, “Processes: A deviation-tolerant approach to software process evolution”, *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting IWPSE '07*, ACM Press, September 2007.