Continuous Process Compliance Using Model Driven Engineering

Fahad Rafique Golra, Fabien Dagnat, Reda Bendraou, Antoine Beugnard

Continuous process compliance using Model Driven Engineering

Fahad R. Golra[1], Fabien Dagnat[1], Reda Bendraou[2], and Antoine Beugnard[1]

[1] IMT Atlantique, IRISA,
Université Bretagne Loire, F-29238 Brest, France
[2] LIP6 / Université Pierre et Marie Curie
Sorbonne Universités, Paris, France

Abstract. Software development methods and standards have existed for decades and the software industry is often expected to follow them, especially when it comes to critical systems. They are of vital importance for establishing a common frame of reference and milestones for software life-cycle planning, development, monitoring and evaluation. However, there is hardly any (semi-)automatic method that ensures the compliance of *de-facto* processes to the adopted *de-jure* standards throughout the development life cycle *i.e.* from specification to enactment. We argue that compliance assurance should be dealt by the process modeling methodologies implicitly to facilitate correct by construction approach for process development. This article presents a framework for modeling software development processes that ensures their continuous compliance to an adopted standard from specification to execution.

1 Introduction

Software development standards define the structure and flow of activities to achieve the objectives efficiently, reduce development risks and promote trust towards external organizations [1]. Compliance of software development processes to these standards is often ensured manually, usually at design time [2]. Different approaches allow translation of design level process models to executable models (*e.g.* [3]). Such approaches can ensure the correctness of a process model for a specific modeling language, but can not guarantee compliance to a process standard. Compliance assessment process is not fully automated because of the way standards are described in a natural language. So it requires considerable human effort to assess compliance to a specific standard. Software development processes are dynamic in nature and often evolve over time. Design time compliance assessment techniques need to be implemented in an active manner, so that all modifications and their repercussions in the process model are evaluated against the standard as the process is being modified.

While some approaches rely on design time assessment of conformance [2], others resort to runtime assessment mechanisms [4,5]. Design time compliance assessment overlooks the possibility of runtime evolution of the processes. Whereas runtime assessment gives us a late feedback on the design of the processes being

used in the organization. Some artifact based process compliance approaches even wait till the availability of the artifacts to give the feedback as to whether the process is compliant or not [6]. We argue that an approach that can continuously guide a process engineer for the development of processes from design time to their enactment can provide a viable solution for the IT industry. It would solve various issues that arise from using multiple approaches of process compliance assessment in different phases of process development life cycle.

A process modeling approach that separates specification phase process models from their corresponding implementation phase models, seems an interesting base for our approach [7]. As a natural extension to this approach, we have added the process enactment capability. A metamodel for executable process models is defined using a bi-layered approach. In this approach, the modeling elements of a single metamodel are partitioned in two conceptual layers *i.e.* abstract and concrete levels. It defines the modeling elements related to process standards at the abstract level and the ones related to process models at the concrete level. Mappings between the two layers are exploited to realize the notion of compliance to standards. The novelty of our approach is to 1) define a methodology that integrates the compliance requirements of the standards with a process model inside a single model, 2) expand the coverage of process compliance from design till enactment phases of process development life cycle and offer it in a single approach, 3) define a methodology where abstractly specified standards can continuously guide the development and enactment of concrete processes.

The rest of this paper is organized as follows. First, we present the key concepts of process compliance and introduce an running example from our case study in Section 2. Then, in Section 3, we explain our process modeling approach. In Section 4, we describe our methodology for continuous process compliance. Then, Section 5 discusses the state of the art in process compliance management. Finally, we conclude this paper in Section 6.

2 Process Compliance

Like all other models in MDE, the language for defining *process models* is defined through metamodels. The primary objective of formally specifying processes is their consistent execution to achieve the intended goals. *Process enactment* is the runtime phase for process models, where humans (or tools) carry out the tasks prescribed in them. *Process trace* records the sequence of activities and the artifacts that were created during their execution. This allows an organization to analyze the runtime behavior of a process to assess its quality and propose any improvement for it, if needed. Different standardization organizations and regulatory bodies define a set of minimum norms that need to be followed so that they can assure that a certain process fulfills a degree of soundness. Compliance to *standards* ensures that processes and practices being followed in an organization are in accordance with adopted/agreed set of norms. These standards can be used either to improve the processes being followed by an organization or to evaluate a specific software provider.

5.10.5 Conducting maintenance reviews

5.10.5.1 Maintenance reviews

a. The maintainer shall conduct joint reviews with the organization authorizing the modification to determine the integrity of the modified system.
EXPECTED OUTPUT: Joint review reports.

5.10.5.2 Baseline for change

a. Upon successful completion of the reviews, a baseline for the change shall be established.
EXPECTED OUTPUT: Baseline for changes.

Fig. 1: Sample activity from ECSS-ST-40C Standard [8]

A software provider may adopt a standard either for improving its internal processes or to assure its clients about the soundness of its processes. This assurance to clients is at times mandatory, specially if working for critical software systems. The norms described in a standard affect the way different tasks are carried out in a compliant organization. In this paper, we use a running example of ECSS-ST-40C [8] that extends a widely adopted ISO/ IEC standard, 12207:2008 [9]. It is a software development standard for space engineering by European Cooperation for Space Standardization. A software sub-contractor working with European Space Agency (ESA) needs to follow ECSS-ST-40C Standard to provide a space mission software. For example, for conducting a maintenance review, ESA's sub-contractor needs to follow the *Conducting maintenance reviews* activity in *software maintenance process* of the ECSS standard (§ 5.10.5 [8]). This activity from the standard, shown in Figure 1, illustrates that the compliant organization is constrained to ensure some minimum requirements. For example, it must 1) perform maintenance reviews (completeness assessment), 2) allocate a person in charge of these reviews who meets a certain criteria of a maintainer (capability assessment), 3) produce specific artifacts like joint review report and baseline for changes (artifact assessment), and 4) ensure that baseline of changes should be produced after the maintenance reviews (flow assessment).

For a software contractor to show that it is compliant to a specific standard, a *process compliance assessment* must be performed by a recognized body. But before presenting itself for assessment, it needs to make sure that its internal processes are actually in compliance with the adopted standard. Software industry needs a process compliance management approach that can handle the processes in different phases of process life cycle i.e. from design to their enactment and even post-enactment analysis based on process trace. This can be carried out using a mixture of forward and backward assessment approaches [10]. Forward assessment is a pre-emptive approach used either before the execution of the process i.e. design time or during the execution. Backward assessment approaches either use the traces produced by the process enactment or rely on the assessment of the produced artifacts against their specifications.

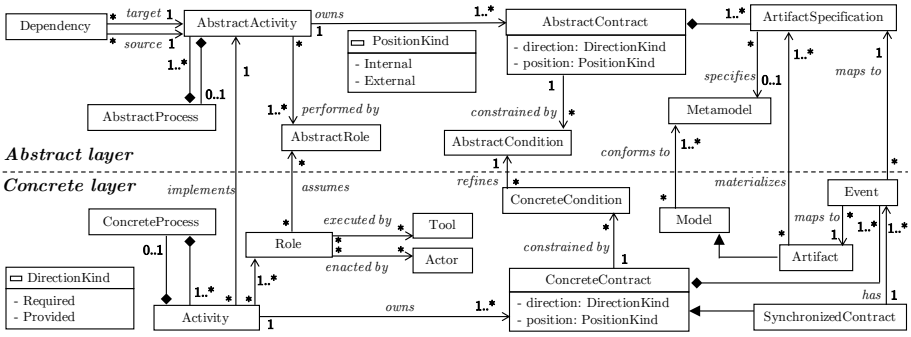


Fig. 2: Fragment of the core metamodel

3 Process Modeling Methodology

We believe that methods and tools should assist process designers to create, manipulate or improve software development processes in a manner that compliance to standards becomes an implicit part of the routine. We have developed a metamodel, that has served as a basis for tool implementations. Figure 2 presents an excerpt from the complete metamodel, which can be consulted here [11]. This metamodel uses two layers of abstraction *i.e.* abstract level and concrete level. The abstract level defines the abstract notions of process design and the concrete level defines the corresponding concrete implementations. It is important to note here that both these layers are conceptual and do not mean that the user needs to develop two different models. This separation of modeling notions in abstract and concrete levels is within a single model. The abstract level suits process standards that are normally defined on the basis of dataflow and do not provide implementations. The user process models are modeled at the concrete level, which provides the modeling notions to deal with concrete implementations of the software development processes. An implementation relationship between the two layers is used to concretize the concepts of compliance. The process model developed using this metamodel can be seen as a single model that captures the structure and behavior of both processes and standards. There is no explicit mapping between the concrete process and the abstract process, because it is implicitly defined through the elements that they contain.

Our metamodel defines a process as an assembly of activities. Processes are inherently hierarchical in nature. This hierarchy is managed through the concept of primitive and composite activities. A composite activity contains a process which gathers a sub-assembly of activities. Whereas a primitive activity specifies or implements the procedure for performing the activity, depending on whether the activity is manual or (semi-)automatic. This hierarchy of activities is defined both at the abstract level and at the concrete level. *Abstract activities* define the higher level specifications for an activity, much like the ones stated in standards. Abstract activities can not be enacted/executed directly, because they lack the necessary implementation details. *Activities* at the concrete level provide complete implementations of the activity, which makes them enactable

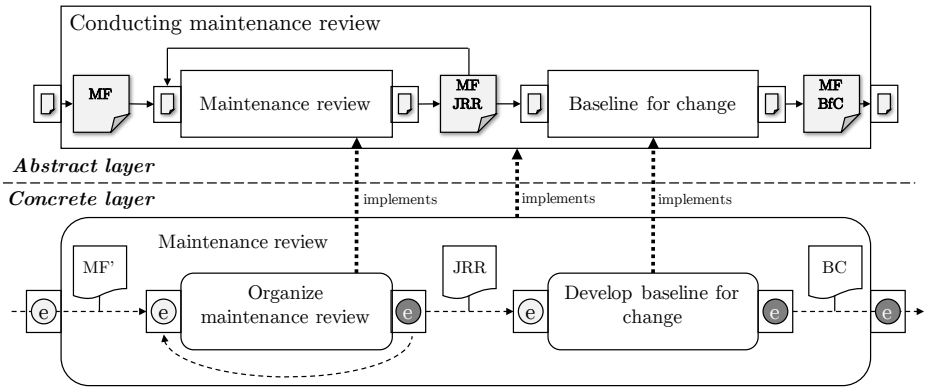


Fig. 3: *Conducting maintenance reviews* activity

(by humans) or executable (by tools). These details pertain to approach, schedule, resource planning, refined milestones, objectives and execution status of the activities during runtime. Figure 3 shows the *conducting maintenance review* abstract activity with both its sub-activities at the abstract level. This activity is implemented by a user process activity, *maintenance review*, at the concrete level. A mapping between an activity of concrete level (*e.g. develop baseline for change*) and the corresponding abstract activity (*baseline for change*) of this process model describes the implementation relationship between them.

Each activity (whether at abstract or concrete level) defines contracts that serve as interfaces for interaction. The notion of *contract* is used to bind the components (activities) using *Design by Contract* [12]. These contracts ensure that all interactions to/from an activity are well-specified and can be monitored. Contracts of an abstract activity precise the artifact specifications that would be needed by the implementing activity to check that they are using/producing the right artifacts at runtime. Required contracts specify artifact specifications that are needed by the abstract activity and provided contracts specify specifications for the artifacts to be produced. It is possible to chose or define a life cycle for each activity. Contracts of the concrete level activities specify the events. A required contract specifies the events that would either trigger an activity or serve as inputs to their life cycle transitions. A concrete provided contract specifies the events produced by the activity during its life cycle. A contract of an activity, at any of these levels, specifies its direction (*i.e.* external or internal) to interact with the activity that contains it or with the activities that it contains.

A notion of *conditions* is also associated with the contracts at both levels. This serves for specifying the pre/post conditions associated with an activity. Defining conditions at both levels allows the refinement of conditions specified at the abstract level. Various software standards specify these conditions for the activities, which can be translated to the conditions at the abstract level. This refinement of conditions at the concrete level provides the possibility to further constrain the interactions of a specific implementation of an abstract activity.

Dependencies between the abstract activities are explicitly defined, because the processes specified in the standards are generally based on the concept of flow between the activities. This flow is usually implicitly defined in standards, where the artifacts produced by one activity are required by the next. We capture this data-flow through the concept of dependencies and artifact specifications provided by the contracts at the abstract level. Some standards are very specific about the artifacts being produced and go as far as pointing to a metamodel, in case the produced artifact is a model. We treat every artifact in our approach as model, whether its metamodel is explicitly defined or not. For example, a document file (*e.g.* docx) does not seem to have a metamodel, but the XML based structure of that document is indeed defined by an implicit metamodel. Activities at the concrete level do not specify this dependency because they are implemented using an event management system. Activities are responsible for creating artifacts at the concrete level, but they specify events in their contracts to notify other activities that they have for example, produced an artifact. We try to capture the control flow of the process during runtime at the concrete level, which is constrained by the data flow specified at the abstract level.

Software standards usually specify a role that is responsible for performing each assigned activity. We translate the role described by the standard as *abstract role* at the abstract level in our metamodel. This abstract role is also refined to the concrete level as *role*. A concrete level role is described by the process model, which depends on the *work breakdown structure* of the compliant organization. Their roles might not be the same as that of a specific standard, because they might be following multiple standards. The role defined in the process model is constrained by the abstract role for its capabilities. This role in the process model can either be played by an actor (human agent) or a tool (software agent), depending on the nature of the activity.

4 Continuous process compliance

We propose continuous process compliance through the use of correct by construction approach that we call *compliance by construction*. It allows to guarantee compliance from the process design time till their monitoring and even during the runtime evolution. To implement this vision, we opted for the development of reference standard models from existing process standards. One important aspect of our methodology is to integrate the modeling elements of this reference process standard in the user process models. A process standard is translated into the abstract level of a process model only once for each standard. This partial process model is then reused multiple times for the development of process models that need to be compliant with this standard.

Each activity in a process has two facets: its static structure and its dynamic behavior. The structure of an activity is defined by its associated roles, properties[3], objectives[3], *etc.*, whereas the behavior through its corresponding states

[3] Concepts not included in the excerpt of metamodel are accessible here [11].

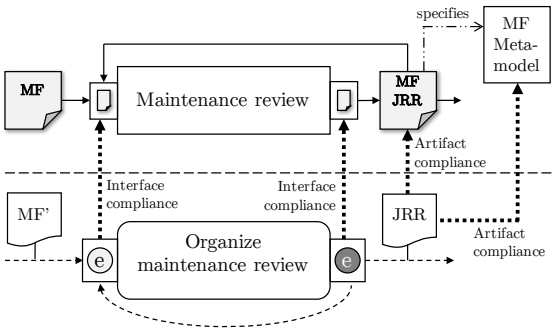


Fig. 4: Contractual compliance

and outputs to various conditions and inputs[3]. The behavior helps in deciding the execution sequence of activities, which may be guided by data (required or produced artifacts) or control (execution dependencies). Continuous process compliance takes both structural and behavioral facets of the process and maps the corresponding notions by associating the elements of the concrete level activities to that of abstract activities (where process standards are specified). The mapping between the modeling elements of the two levels is carried out at the design time. While the abstract level of the model remains the same, the concrete level of the process model is transformed to get the runtime process model used for process enactment. During process enactment, the mapping between the two levels help in ensuring runtime compliance to process standards.

4.1 Design-time compliance management

A process model is developed by reusing an already developed partial process model from an adopted process standard at the abstract level (§ 3). This partial process model is then enriched with the defined processes of the compliant organization. The constructs of the standard model are accessible to this process model during the development, which are used for developing a compliant process. This form of compliance, ensured at the development phase of process life cycle, is called the design-time compliance management in our methodology. For design-time compliance, we exploit the mappings between the concrete and abstract levels of our process metamodel to perform consistent checks concerning compliance of contracts, capacity and hierarchy of activities.

Contractual compliance: For making an activity compliant to its corresponding abstract activity, we need to ensure compliance between their contracts. A contract of abstract activity consists of artifact specifications and a contract of concrete level activity contains events (§ 3). The contractual compliance between them is ensured using following assessments.

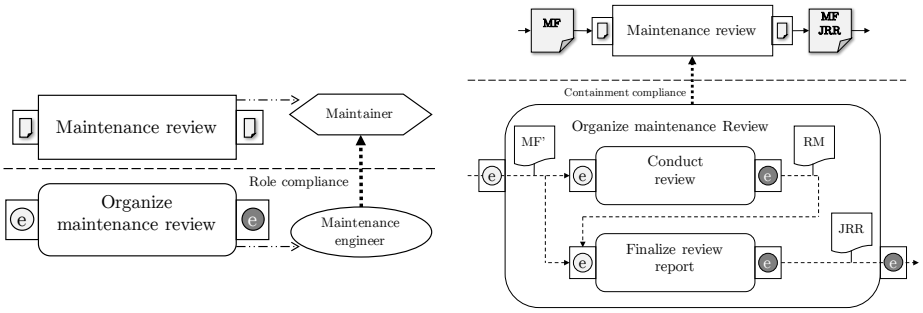
– *Interface assessment:* Generally, the standards define the input and output artifacts of the activities that are placed in a sequence/flow such that a later activity uses the artifact(s) produced by the former activities. For standards

translated at the abstract level of the model, these interactions between the abstract activities are realized through artifact specifications. However the activities from the user process model either produce or listen to events, through their contracts. So the compliance of interfaces for an activity is to ensure that the events produced and listened by it are compliant to the artifact specifications of the corresponding abstract activity. A state machine related to each artifact specification is available in the abstract contract. In our example, shown in Figure 4, the *maintenance review* abstract activity from the standard is implemented by *organize maintenance review* activity in the user process model. The required contract of the concrete level activity listens to the events related to *maintenance file*. The process designer may choose to trigger this activity with an event that confirms the availability of maintenance file. The execution of this activity should produce the *joint review report* at runtime. On creation of this artifact, *organize maintenance review* activity fires an event, which could be used by subsequent activities. In this scenario, the process designer should make sure that this activity would listen for an event compliant to *maintenance file* and produce an event compliant to *joint review report*. Following checks are needed to ensure the contractual compliance between the concrete and abstract activities.

1. The *required events* of the concrete activity are a subset of events specified in the required artifact state machine.
2. The events specified in the provided artifact state machine are a subset of the *provided events* of the concrete activity.

– *Artifact assessment:* Standards define the input and output artifacts of their activities through artifact specifications. This difference between the artifact and its specification is important in our methodology. The artifact specification is modeled at the abstract level, whereas the actual artifact at the concrete level. We consider each artifact as a model that conforms to its metamodel. Sometimes the metamodel of an artifact is implicit. When the metamodel is explicit, the artifact specification points to it. In such a case, (artifact) model can be checked against the metamodel to verify its structure and properties, using existing model checking techniques. In our example, as shown in Figure 4, the *joint review report* produced by the *organize maintenace review* activity is checked against its specification (*MF JRR*). The standard does not provide a metamodel in our example, but in case it did, *joint review report* would have to be checked against it as well.

Capability compliance: Activities specified in the standard are associated to the abstract roles that perform them. An abstract role is a set of capabilities that are required from the person, team or tool performing a specific activity. Software vendors normally define their own roles, according to their particular team structures. The mapping between the role at concrete level and the abstract role is translated as a *responsibility assignment matrix (RAM)*. This matrix ensures that the concrete level role complies with all the capability requirements specified by the standard. In our example, the ECSS standard defines a *maintainer*



(a) Capability compliance (b) Containment compliance

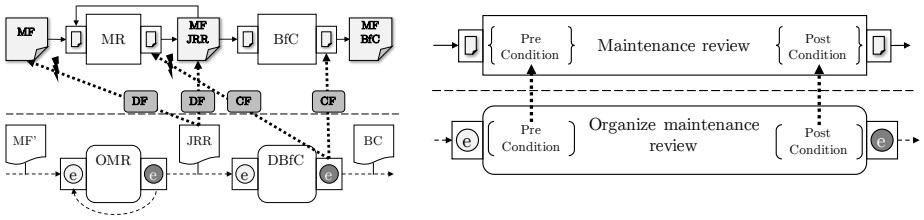
Fig. 5: Capability and containment

abstract role, as shown in Figure 5a. The concrete role that is responsible for performing the compliant activity is *maintenance engineer*. The mapping between the maintenance engineer and the maintainer is translated to the RAM, which maps each capability of the concrete level role to the abstract role.

Containment compliance: Each activity realizes a given abstract activity. A process designer adds the implementation details to an activity during its development while maintaining its compliance to the corresponding abstract activity. Adding these implementations can either involve enriching the activity directly or making it a composite activity, hence adding further activities deep in its hierarchy. The milestones set by an abstract activity are further refined in the concrete level activity. This refinement can introduce intermediate goals that can be set as objectives for the sub-activities, when implemented as a composite activity. Figure 5b shows the *maintenance review* abstract activity from our example. It is implemented by *organize maintenance review* composite activity containing two sub-activities: *conduct review* and *finalize review report*. For *organize maintenance review* activity to remain compliant to its abstract activity, its sub-activities have to respect the compliance as well. Our metamodel (§ 3) defines a *position* and a *direction* for every contract. *Organize maintenance review* activity listens to the required events from its *external required contract*. Once an event triggers the activity, it is passed on to the sub-activities through its *internal provided contract*. The sub-activities produce the planned artifacts and fire the concerned events, which are moved up in the hierarchy in the same fashion. Containment compliance ensures both the contractual and capability compliance for the contained activities. Contractual compliance is assured when:

1. The events required by the sub-activities are a subset of events provided by the *internal provided contract* of the parent activity.
2. The events required by the *internal required contract* of the parent activity are a subset of events fired by the sub-activities.

Capability compliance is assured in the hierarchy, by automatically associating the role of parent activity to the roles of sub-activities. It is important to note



(a) Flow compliance (b) Condition compliance

Fig. 6: Flow and condition compliance

that multiple roles can be associated with an activity. The role who performs an activity can be different from the role who supervises it.

4.2 Runtime compliance management

Runtime assurance of compliance depends on the state of the user defined process during its enactment. The state of a process is defined by the collective state of all the activities that it contains. The state of a particular activity depends on its defined state machine and the events that it has consumed at a particular time. When an activity changes its state, it can fire events that can be consumed by other activities. In this event-based enactment paradigm for the processes, the compliance of a user process to a standard at runtime is assessed though the compliance of flow, conditions, traceability, completeness and capacity.

Flow compliance: Processes are defined as a (partially) ordered set of activities. The order of activities is due to the dependence of certain activities over others, which comes from the handshake of data, artifacts or control. The order in which the activities are enacted in a user process model needs to conform to the standard. A compliant order of enactment for the activities is ensured through the runtime assessment of data-flow and control-flow of the process.

- *Data-flow assessment:* In a process standard, activities are defined in a sequence such that the artifacts produced by an activity are required by the following activities. This dependence of one activity over another, based on the artifacts, is captured at the abstract level of our process model through the notion of *contract* and *dependency*. The contracts at the abstract level of the process model require and provide *artifact specifications*. When an abstract activity (of standard process) requires the artifact specification that is produced by another, this dependence is explicitly stated by the use of *dependency* (§ 3). However, the user process model uses an event driven paradigm for enactment. Events at the concrete level map to the artifact specifications. A data-flow dependence between two activities translates to the events through the mappings between events and artifact specifications. In our example, shown in Figure 6a, once the *organize maintenance review (OMR)* activity starts its execution, event related to *joint review report (JRR)* can be fired. However, events that map to *maintenance file (MF)* can not be

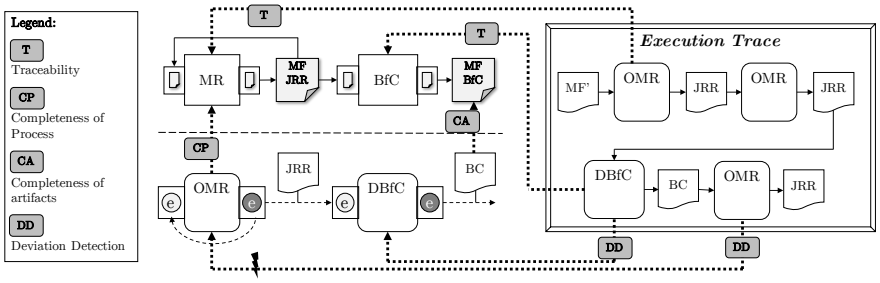


Fig. 7: Traceability and Completeness compliance

fired anymore. OMR activity allows multiple iterations, but the subsequent executions also require events related to JRR.

– *Control-flow assessment:* Some activities are only meant to execute some operations without creating an artifact. For such activities, if their order of execution is not constrained by the standard, the event-driven paradigm allows a reactive mechanism to order them according to the execution state of the process. In case, they are constrained by the standard, compliance becomes mandatory. In this case, the events use the notion of *dependency* at the abstract level to order the execution sequence. In our example, in Figure 6a, the dependence of *develop baseline for change (DBfC)* activity on OMR is dictated by data-flow, however the control-flow decides the number of iterations for OMR and subsequent transfer of control to DBfC.

Conditions compliance: Apart from specifying the input and output artifacts of an activity, process standards can also constrain them by specifying pre and post conditions. These conditions are translated into the *abstract condition* at the abstract level of our metamodel[4]. Conditions specified at the concrete level of the model are the refinement of the abstract conditions. The mapping between the conditions of *organize maintenance review* and *maintenance review* shows their refinement in Figure 6b. For the user process model to be compliant with the standard, all abstract conditions should be implemented at the concrete level. Concrete conditions may further constrain the user activities based on their specific implementation details, however they can not relax the conditions.
Traceability compliance: The runtime compliance management uses the execution trace of the process model for the following assessments.

– *Traceability assessment:* Contrary to other approaches, our approach incorporates the modeling elements of the standards at the abstract level of the model and the user process maps to them at design time. Modeling elements of the standard don't execute at runtime, however user process activities can trace back to them. This allows to evolve the user process in a compliant manner, even during the runtime. In our example, shown in Figure 7, a traceability link is maintained between *OMR* and *MR*. During the execution of

[4] Conditions are further refined into pre and post conditions in the metamodel [11]

the process, *OMR* activity can be replaced by *OMR'*. However, for the user process to stay compliant, *OMR'* needs to follow the design time compliance assessments *i.e.* contractual, capability and containment compliance.

- *Deviation Detection:* The traceability links allow to map the runtime process to the adopted standard. The runtime order of execution for the activities is generated using a set of execution constraints from the specific runtime conditions (execution history of activities through process trace) and the defined dependencies. A constraint analyzer detects the any violated constraints if the process enactment deviates from the defined process. In our example, shown in Figure 7, lets imagine a case where the project manager wants to re-enact OMR after the execution of DBfC, because he is not satisfied with the results. This is contrary to the defined process. A violated constraint in this case may put the compliance of runtime user process to the adopted standard at risk. Thus runtime deviation detection triggers the design-time compliance assessment for the modified part of the process before the actual execution of the forced deviation. Then it helps user to pinpoint the exact conflicts by stating which constraints will be violated by this specific user decision. In this case, the user is notified that the order of execution of OMR and DBfC is against the adopted standard.

Completeness compliance: Compliance to a standard is not ensured, unless the user process guarantees to execute all the activities defined by the standard. The mappings between the user process elements and the elements of the standard, established at design-time, help ensure the completeness of compliance at runtime. Figure 7 shows the mappings from the activities (*OMR*) to the abstract activities (*MR*) and artifacts (*BC*) to the artifact specifications (*MF BfC*). These mappings are used for a continual runtime assessment for process enactment. For a compliant user process, it needs to guarantee that at least one concrete activity is enacted for each abstract activity of the process standard.

Capacity compliance: *Capability compliance* at design time checks the mapping between the concrete level *role* and the *abstract role*. Role at the concrete level is enacted by an *actor* for manual activities, executed by a *tool* for automatic activities and by both for semi-automatic activities. During the runtime, the *responsibility assignment matrix* of capability compliance is reused to map the competences of actors and tools to the corresponding role. These mappings are then used to ensure capacity compliance, such that the competences constrained by the standard are fulfilled by the actors and tools. This compliance further helps in implementing the concrete conditions related to the roles *e.g.* two activities X and Y can not be enacted by the same maintenance engineer.

5 Related Work

With process standards becoming increasingly popular as a mean to guarantee the quality of a software deliverable, we see multiple approaches that deal with the challenges of compliance assessment for user processes to the adopted standards. We classify these approaches in three categories: rule-based, artifact-based

and reference model based approaches. Rule based approaches include multiple proposals for formal modeling of business rules both by academia (*e.g.* [5,10]) and industry (e.g. ILOG by IBM). A closely related approach models control objectives for monitoring the execution behavior of the user processes [2]. These approaches focus on the backward assessment of the process model, hence even if they detect noncompliance in some part of the process, that part needs to be re-modeled and re-enacted. They do not offer any support during the initial development phases of the process models.

Out of different approaches that assure compliance management for process models, there are some that offer artifact based compliance [6]. They model the deliverables expected from the activities by a standard. Then the artifacts developed by the user process model are verified against the expected deliverables (artifact specifications). Just like rule-based approaches, the problem is that the artifacts are produced late in a project. In case of noncompliance, the process needs to be modified and considerable effort of process design and execution is wasted. Having the compliance assured at design time, we use this kind of compliance as a secondary assessment method for the quality control of the artifacts. Reference model based approaches develop a reference model from the adopted standard [13,14]. They use different model checking approaches for assessing the compliance of the user process against the developed reference model. These approaches are closest to our methodology, as we also model the constructs of a given software process standard. However, we do not translate the standard as a different model, we put its constructs within the process model at an abstract level. This allows us to support compliance not only in design and development phases, but all along the process development life cycle.

A limitation of our approach is that modeling elements of the abstract model (process standard) becomes part of the user process model. Even though it offers the benefits of active compliance assessment, it makes the process model 'heavier'. It might seem as combining the concepts from both process standard and the user process in a single model might make the development of process models even more complex. Actually, an already modeled process standard serves as a partial model for developing any process model that needs to comply with that standard. Our prototype guides the user through process development using the modeling elements of the process standard. Hence the effort for the development of a process model is in fact reduced.

6 Conclusion

We have presented a model driven approach to continuous process compliance for a complete coverage of process development life cycle. We proposed a methodology for modeling the constructs of a user process model and a given standard in a single process model. Constructs from the user process and the standard are modeled in two different levels within this model. We create mappings between these two abstraction levels and use them to ensure compliance of the user process model to an adopted standard. This compliance is assessed both at

design time and at the runtime. At design time, we concentrate on the structural elements of the process model, whereas at runtime, we focus on constraining its execution behavior according to the compliance requirements of the standard. Hence we provide an overall methodology for the development of process models using compliance by construction and then verify the compliance during the runtime. The current prototype implementation of our methodology supports a single standard for the moment. We are working towards the compliance of a user process model to multiple standards simultaneously. Our vision is to provide a methodology where software vendors can define their processes in an intuitive way and compliance to the quality standards becomes part of this routine.

References

1. Wüllenweber, K., Beimborn, D., Weitzel, T., König, W.: The impact of process standardization on business process outsourcing success. Information Systems Frontiers **10**(2) (2008) 211–224
2. Sadiq, S., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: Business process management. Springer (2007) 149–164
3. Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.: Translating standard process models to BPEL. In: Advanced Information Systems Engineering, Springer (2006) 417–432
4. El Kharbili, M., Stein, S., Pulvermüller, E.: Policy-Based Semantic Compliance Checking for Business Process Management. In: MobIS Workshops. Volume 420., Citeseer (2008) 178–192
5. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes based on Monitoring Real Behavior. Information Systems **33**(1) (2008) 64–95
6. Emmerich, W., Finkelstein, A., Montangero, C., Antonelli, S., Armitage, S., Stevens, R.: Managing standards compliance. IEEE Transactions on Software Engineering **25**(6) (Nov 1999) 836–851
7. Golra, F.R., Dagnat, F.: Generation of dynamic process models for multi-metamodel applications. In: 2012 International Conference on Software and System Process (ICSSP), IEEE (June 2012) 48–57
8. ECSS, Requirements & Standards Division: Space Engineering - Software, ECSS-E-ST-40C (2009)
9. ISO/IEC: Systems and Software Engineering - Software Life Cycle Processes, ISO/IEC 12207, IEEE Std 12207-2008 (2008)
10. El Kharbili, M., Stein, S., Markovic, I., Pulvermüller, E.: Towards a framework for semantic business process compliance management. Proceedings of GRCIS (2008)
11. Golra, F.R.: A Refinement based methodology for software process modeling. PhD thesis, Télécom Bretagne, Université de Rennes 1 (2014)
12. Meyer, B.: Applying 'design by contract'. Computer **25**(10) (1992) 40–51
13. Chung, P.W., Cheung, L.Y., Machin, C.H.: Compliance Flow - Managing the compliance of dynamic and complex processes . Knowledge-Based Systems **21**(4) (2008) 332 – 354
14. Panesar-Walawege, R., Sabetzadeh, M., Briand, L.: A Model-Driven Engineering Approach to Support the Verification of Compliance to Safety Standards. In: 22nd International Symposium on Software Reliability Engineering (ISSRE). (Nov 2011) 30–39