Formalizing a Policy-based Compliance Control Solution with Event-B

Laura González and Raúl Ruggia
Instituto de Computación, Facultad de Ingeniería, Universidad de la República,
J. Herrera y Reissig 565, Montevideo, Uruguay

Keywords: Policy-based Management, Compliance Management, Event-B, Rodin Platform.

Abstract: Compliance management is gaining increasing interest in inter-organizational service-oriented systems, which are usually supported by integration platforms. Due to their mediation role and capabilities, these platforms constitute a convenient infrastructure for controlling compliance requirements affecting message exchanges between organizations. This paper proposes a formal model for a policy-based compliance control solution introduced in our previous work for such platforms. The model, which was developed using Event-B, provides unambiguous specifications and enables formal proofs as well as the verification of the solution operation.

1 INTRODUCTION

Compliance management is gaining increasing interest in inter-organizational service-oriented systems. It aims to ensure that organizations act in accordance with multiple established regulations (e.g. standards, laws) (Tran et al., 2012), which introduce compliance requirements. These requirements may affect inter-organizational message exchanges and may concern different areas (e.g. data quality, data protection).

Controlling compliance requirements (i.e. assessing their fulfillment and acting accordingly) is a major issue in these scenarios (Elgammal et al., 2016). Organizations are thus required to develop compliance solutions in order to control such requirements.

In turn, inter-organizational service-oriented systems can be supported by integration platforms, which are specialized infrastructures providing capabilities to facilitate the integration of heterogeneous systems. This way, systems in different organizations communicate with each other by invoking services through the platform via message exchanges, which may be processed by integration flows to solve heterogeneity issues. These flows leverage different integration mechanisms (e.g. message transformation).

Due to their mediation role and capabilities, integration platforms constitute a convenient infrastructure for controlling compliance requirements that affect inter-organizational message exchanges (González and Ruggia, 2018b). For example, a transformation may remove sensitive data from messages in order to comply with data protection regulations.

In our previous work we proposed an approach to compliance management within inter-organizational service integration platforms (González and Ruggia, 2018a). The main elements of the proposal are a life cycle, a conceptual framework, and a compliance management system. This system comprises a system-level compliance control (SCC) subsystem and a policy language (González and Ruggia, 2018c). The language provides the means to specify how requirements have to be controlled using the components of the SCC subsystem. The SCC subsystem is responsible for controlling compliance at the system-level by processing all messages exchanged through the platform based on the deployed policies. This control may lead to compliance actions (e.g. remove data from messages) which are based on integration platforms mechanisms (e.g. message transformation).

In this context, formally assessing the behavior of integration platforms and compliance control systems appears paramount to enhance the reliability of their operations. Nevertheless, unlike research on the architectural and technological aspects of such platforms, formalization proposals are still very scarce.

This paper constitutes a step forward in our compliance management approach by proposing a formal model of the SCC subsystem, using the Event-B modeling method and the Rodin platform (Abrial, 2010). The formalization provides unambiguous specifications of how messages are processed by this subsystem according to policy language constructs. It also enables formal proofs of safety properties and the verification of the SCC subsystem in different scenarios.

González, L. and Ruggia, R.
Formalizing a Policy-based Compliance Control Solution with Event-B.
DOI: 10.5220/0008120406050613
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 605-613
ISBN: 978-989-758-379-7

The rest of the paper is organized as follows. Section 2 presents a motivational scenario and background on Event-B. Section 3 describes the policy-based compliance solution proposed in our previous work and Section 4 presents its formalization using Event-B. Section 5 analyzes related work. Finally, Section 6 presents conclusions and future work.

2 PRELIMINARIES

This section presents a motivational scenario as well as background on Event-B and the Rodin Platform.

2.1 Motivational Scenario

The scenario is inspired by the Uruguayan e-Government Interoperability Platform (egovIP) (González et al., 2012), which uses a general purpose integration platform. The egovIP enables and facilitates government organizations to offer business services leveraging the web services technology. These web services, which are usually hosted on organizations' infrastructure, are exposed and invoked through proxy services deployed on the egovIP. The platform is thus able to process all service invocations and apply mediation operations to them.

For example, as shown in Figure 1, the Ministry of Public Health (Ministerio de Salud Pública, MSP) offers the Death Certificates Service to other organizations within the platform. In particular, this service has an operation (i.e. getCertificate), which receives a National Identification Number (NIN) and returns death certificate data of the specified citizen[1].

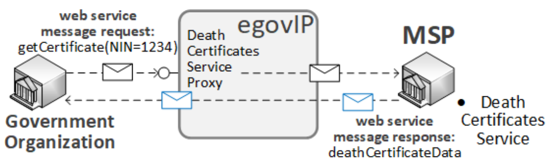

Figure 1: egovIP - Death Certificates Service.

In this context, compliance requirements arise in various areas such as Quality of Service (e.g. compliance with SOAP 1.1), Data Quality (e.g. syntactic accuracy) and Data Protection (e.g. privacy issues). In particular, syntactic accuracy requirements state that countries should be specified using the Alpha-3 codes of the ISO 3166-1 standard.

[1]https://www.agesic.gub.uy/innovaportal/v/5333/9/agesic/consulta-certificado-de-defuncion-electronico.html

2.2 Event-B and Rodin Platform

Event-B is a modeling method for formalizing and developing systems that can be modelled as discrete transition systems (Romanovsky and Thomas, 2013). The method is centered around the notion of events (i.e. transitions) and its main purpose is to help engineers developing systems that will be correct by construction (Abrial, 2018). The basis for the models in Event-B is first-order logic and a typed set theory.

The models described with Event-B are built by means of: i) contexts, which contain the static part of the system, and ii) machines, which contain the dynamic part of the system (Abrial et al., 2010). Event-B supports the incremental development of models by context extension and machine refinement. This provides the means to start from a very abstract model and to subsequently introduce requirements in model refinements. Verification of model properties is performed by doing formal proofs of theorems, which are automatically generated by the Rodin platform and may be proved automatically or interactively leveraging this platform (Su et al., 2014).

A *context* may contain carrier sets (i.e. user-defined types), constants (i.e. static objects), axioms (i.e. properties of carrier sets and constants) and theorems (i.e. derived properties of carrier sets and constants) (Romanovsky and Thomas, 2013).

Machines specify behavioural properties of Event-B models and may connect with several contexts (Romanovsky and Thomas, 2013). Machines may contain variables (which define the state of a machine), invariants (which constraint variables), theorems (i.e. additional properties of the variables derivable from invariants), events (which may change the state of the machine) and a variant (which may be used to prove convergence properties). Machine refinement provides a mean to introduce more details about the dynamic properties of a model.

Events describe the dynamics of machines (Romanovsky and Thomas, 2013). They may contain parameters, guards (which specify the conditions under which an event is enabled) and actions (which describe how the state variables evolve when the event occurs). The simplest form of an event action is a simple deterministic assignment (Su et al., 2014). A set of events can be proved to collectively converge (i.e. these events cannot take control forever).

Proof obligations specify what is to be proved for an Event-B model (Abrial, 2010). They are automatically generated by the Rodin platform and transmitted to the provers. The relevant proof obligations rules for this work are invariant preservation, numeric variant, variant and well-definedness.

3 COMPLIANCE SOLUTION

This section describes the policy-based solution to compliance control proposed in our previous work (González and Ruggia, 2018c). The solution comprises: i) a system-level compliance control (SCC) subsystem, which extends integration platforms in order to control compliance at the message level, and ii) a compliance policy language, which enables the specification of policy-based compliance methods in order to state how requirements have to be controlled using the components of this subsystem.

3.1 SCC Subsystem

The SCC subsystem is part of the Compliance Control Subsystem proposed in our compliance management approach. It processes all platform messages to control compliance at the system-level (i.e. message-level), based on the deployed compliance methods. This control may lead to system-level compliance actions (e.g. remove data from messages).

The SCC subsystem follows a policy-based approach to control compliance. It has the typical architecture of policy-based solutions and it is inspired by the components and interactions specified in XACML (OASIS, 2013). Its main components are: a Compliance Policy Enforcement Point (cPEP), a Compliance Policy Decision Point (cPDP), a Compliance External Services Point (cESP), a Compliance Event Monitoring Point (cEMP), a Compliance Logging Service (cLS) and a Compliance Actions Service (cAS).

The cPEP is responsible for processing messages and enforcing compliance according to the decisions made by the cPDP (e.g. the cPEP may accept or reject messages). The cPDP renders compliance decisions based on policies deployed in the platform and data included in cPEP requests. The cPDP may also interact with the cESP and the cEMP in order to get the external services responses and monitored events, respectively, required to make decisions.

When the cPEP receives a compliance response (e.g. accept, reject) from the cPDP, it leverages different components in order to enforce compliance. In particular, it may send monitoring data to the cEMP, it may interact with the cESP in order to perform an asynchronous external service request, it may log compliance data leveraging the cLS component, it may send these data to the BCC subsystem, it may interact with the cAS in order to request the execution of adaptation actions (e.g. filtering messages content) and compensation actions (e.g. executing a compensation service), and it may deliver the message to other components of the platform for later processing.

Event-driven interactions may be also carried out within this subsystem when the cPDP receives a monitored event from the cEMP or an asynchronous external service response from the cESP. In both cases, the cPDP performs a similar processing as it does with compliance requests. This processing may generate a compliance event which is sent from the cPDP to the cPEP and it is processed by the cPEP in a similar way as it does with compliance responses.

The SCC subsystem supports seven decision values including: *accept* (the message exchange is compliant and it must be accepted), *reject* (the message exchange is not compliant and it must be rejected), *allow* (the message exchange is not compliant but it must be allowed), *repair* (the message exchange is not compliant and actions must be taken to try to make it compliant; after that, the message exchange has to be evaluated again), and *verify* (the message exchange is going to be verified to check if it is compliant or not; in the meantime, it has to be allowed).

Figure 2 presents a sequence diagram showing the decision and enforcement processes that are carried out within the SCC subsystem, when controlling a Data Quality (DQ) requirement of the motivational scenario with a policy-based compliance method.

The DQ requirement specifies that the ”country” returned by the operation getCertificate has to be compliant with Alpha-3 codes. The compliance method to control this requirement comprises two policies. PolicyForResponses applies to responses returned by the operation, it states the value *verify* as the result and it states an interaction with the cESP in order to check if a country value is compliant with Alpha-3 codes. PolicyForExternalResponses applies to responses returned by the cESP and it states *allow* with a compensation action if the cESP returns false (i.e. the country is not compliant with Alpha-3 codes). Otherwise it states *accept* as the result.

The decision and enforcement processes to control the DQ requirement are described as follows. When a compliance request is received, the cPDP returns *verify* as the compliance decision and specifies an external request to be performed by the cPEP, according to PolicyForResponses. An enforcement process takes place which delivers the message and performs the request to the cESP. Later, the cESP returns a response to the cPDP which indicates if the country included in the request is compliant with Alpha-3. According to PolicyForExternalResponses, the cPDP sends a compliance event to the cPEP with *accept* or *allow* as the decision. In the later case, a compensation action is also included in the event. Finally, a second enforcement process takes place where the compensation is performed if the country is not compliant.

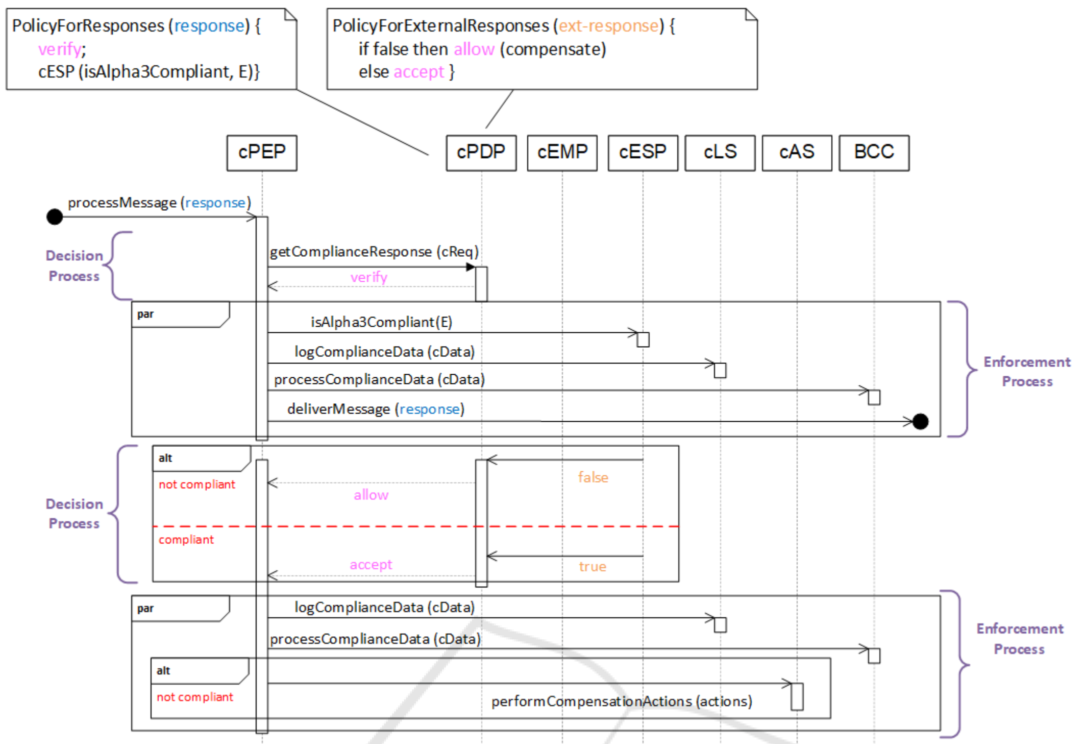
PolicyForResponses (response) {
verify;
cESP (isAlpha3Compliant, E)}
PolicyForExternalResponses (ext-response) {
if false then allow (compensate)
else accept }
cPEP
cPDP
cEMP
cESP
cLS
cAS
BCC
processMessage (response)
Decision Process
getComplianceResponse (cReq)
verify
par
isAlpha3Compliant(E)
logComplianceData (cData)
processComplianceData (cData)
deliverMessage (response)
Enforcement Process
alt
not compliant
allow
false
Decision Process
compliant
accept
true
par
logComplianceData (cData)
processComplianceData (cData)
Enforcement Process
alt
not compliant
performCompensationActions (actions)

Figure 2: SCC Subsystem - Controlling a Data Quality Requirement.

3.2 Compliance Policy Language

The Policy Language for Compliance (PL4C) is geared towards enabling the specification of policy-based compliance methods that indicate how compliance requirements have to be controlled at the message level by the components of the SCC subsystem. PL4C is inspired by XACML (OASIS, 2013) and FACPL (Margheri et al., 2017).

The abstract syntax of PL4C is specified by an Ecore metamodel and restrictions for model elements. The concrete syntax of PL4C was developed with Xtext and it is specified using the Xtext grammar language. Listing 1 presents the PL4C specification of the compliance method presented in Section 3.1.

First, Lines 1-8 specify general properties of the method. Lines 2-4 establish that the method controls the CountryCompliantAlpha3 requirement over the GetCertificate operation. Lines 5-7 specify that the method is applicable to messages that are returned by this operation (of the Death Certificates service).

Second, lines 10-14 declare the use of an external functionality (i.e. IsAlpha3Compliant) and a compensation action mechanism (i.e. LogCompliance).

Then, Lines 16-28 specify a message policy which has the same scope of the method (lines 17-19) and states the result *verify* (line 23). The policy also specifies an ExternalRequestAction (lines 25-28) in order to invoke the external functionality isAlpha3Compliant in an asynchronous way.

Finally, Lines 29-44 specify an external response policy which applies to the response obtained from the IsAlpha3Compliant functionality (line 30). This policy has a rule (lines 34-37) which is evaluated to *allow* if the external functionality returns a false value. Otherwise, it is evaluated to *accept* as specified in the combining algorithm (lines 31-32). The rule also specifies a CompensationAction which logs the non-compliance when the decision is *allow*.

4 EVENT-B FORMALIZATION

This section presents a formalization of the SCC subsystem developed with Event-B (Abrial, 2010) and the Rodin platform (Abrial et al., 2010).

4.1 Formalization Approach

The formal specification of the SCC subsystem aims to formally specify the execution of the decision and enforcement processes, which are carried out within this subsystem according to the different PL4C constructs. The overall formalization strategy consists in:

- Modeling the general operation of the SCC subsystem independently of the deployed compliance

Listing 1: Compliance Method for DQ Requirement.

```
ComplianceMethod DQ_CountryAlpha3_GetCertificate
 controls:
   requirement ”CountryCompliantAlpha3”
   over object ”GetCertificate”
 scope:
   and (fromService ”DeathCertificates”,
       fromOperation ”GetCertificate”)
 algorithm:
   accept UnlessOther
 uses:
   ExternalFunctionality(”IsAlpha3Compliant”)
     isAlpha3Compliant
   CompensationActionMechanism (”LogCompliance”)
     logCompliance
 policies:
   MessagePolicy PolicyForResponses
     scope:
        and (fromService ”DeathCertificates”,
            fromOperation ”GetCertificate”)
     algorithm:
        accept UnlessOther
     rules:
        Rule RuleForResponses (verify)
     actions:
        ExternalRequestAction (verify, allow)
           isAlpha3Compliant ([”country”,
             getDataItem(”//country”,
                this.msgSrvMessage)])
   ExternalResponsePolicy PolicyForExternalResponses
     onResponse: isAlpha3Compliant
     algorithm:
        accept UnlessOther
     rules:
        Rule ”Rule1” ( allow
          condition:
             not(boolean(
               response.getResponseValue(”result”)))
     actions:
        CompensationAction (allow) logCompliance
           ([”certNumber”,
             getDataItem (”certNumber”,
               response.message.msgSrvMessage)],
            [”country”,getDataItem (”country”,
               response.message.msgSrvMessage)]))
```

elements (e.g. methods, policies, mechanisms).

- Leveraging Event-B elements (e.g. predicates, guards) to specify how PL4C constructs (e.g. policies) extend the operation of the subsystem.

This way, the Event-B formalization consists of a core model, which does not depend on the deployed elements (e.g. methods), and extensions, which address the operation of the subsystem according to the deployed elements (e.g. to address DQ requirements).

In turn, the refinement strategy consists in:

- Specifying an initial model with a high level description of the operation of the SCC subsystem.
- Developing refinements which provide more details on a specific part of this operation.

Table 1 presents the aspects that are addressed by the initial model and some of the refinements.

Table 1: Initial Model and Refinements.

Model	Description
Initial Model	High level processing (e.g. of messages, external responses).
2nd Refinement	Getting applicable methods.
5th Refinement	Evaluating rules.
8th Refinement	Making a compliance decision.
12th Refinement	Enforcing compliance.

4.2 Initial Model

The initial model of the SCC subsystem addresses the high level processing of messages, monitored events and external responses. The UML State Machine diagram in Figure 3 presents part of this processing. After initialization, the SCC subsystem waits for data (e.g. messages). When a message is received, a compliance request is built and processed. This generates a compliance response which is also processed by the subsystem. Afterwards, the processing of the message may finish or the message may have to be processed again (e.g. after repair actions).

The initial model comprises two contexts and one machine. The contexts define an enumerated set to represent the different states of the subsystem (e.g. WaitingForData), a set of positive natural numbers which represents the messages waiting to be processed, and a natural number indicating the maximum number of retries when re-evaluating a message.

The machine, presented in Listing 2, specifies eleven variables which define its state. In particular, the machine defines variables for the message that is being processed (i.e. message), the messages received by the subsystem (i.e. receivedMessages), and the current state of the subsystem (i.e. state), among others. A variant is also defined to prove convergence of an event which is presented in what follows.

The machine also defines seventeen events: one for each transition between states. Listing 3 presents the definition of the ReceiveMessage and ReprocessMessage events. Note that the ReprocessMessage event is defined as ”convergent” in order to, along with the specified variant, prove its convergence.

The proof obligation generator of the Rodin platform produced twenty-six proof obligations which were all discharged automatically. In particular, twenty-four invariant preservation proof obligations, one numeric variant proof obligation and one variant proof obligation were generated and discharged.

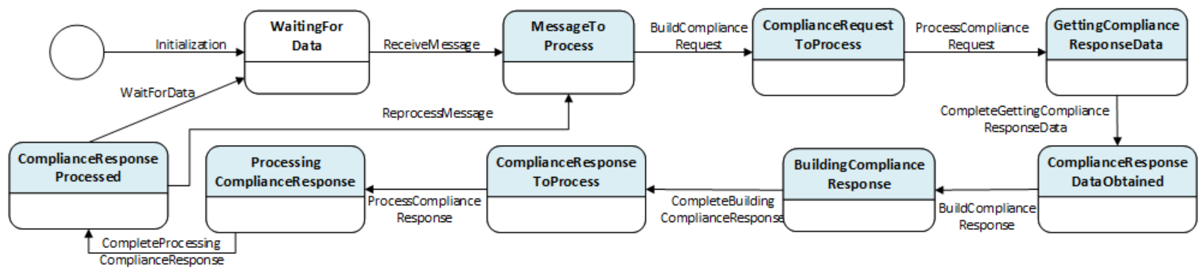
Initialization
WaitingFor Data
ReceiveMessage
MessageTo Process
BuildCompliance Request
ComplianceRequest ToProcess
ProcessCompliance Request
GettingCompliance ResponseData
WaitForData
ReprocessMessage
CompleteGettingCompliance ResponseData
ComplianceResponse Processed
Processing ComplianceResponse
ComplianceResponse ToProcess
BuildingCompliance Response
ComplianceResponse DataObtained
ProcessCompliance Response
CompleteBuilding ComplianceResponse
BuildCompliance Response
CompleteProcessing ComplianceResponse

Figure 3: Initial Model - High Level Processing.

Listing 2: Initial Model - Machine scc00.

```
MACHINE
    scc00
SEES
    ctx00b
VARIABLES
    message // message that is being processed
    receivedMessages // messages received by the SCC subsystem
    state // current state (e.g. stWaitingForData)
    nextMessage // next message to be processed
    complianceRequest // current compliance request
    nextComplianceRequest // next compliance request to be built
    complianceRequests // generated compliance requests
    complianceResponse // current compliance response
    nextComplianceResponse // next compliance response to be built
    complianceResponses // generated compliance responses
    retries // current number of retries
INVARIANTS
    inv1 : message ∈ messages ∪ {0}
    inv2 : receivedMessages ⊆ messages
    inv3 : state ∈ PROCESSING_STATE
    inv4 : nextMessage ∈ ℕ₁
    inv5 : complianceRequest ∈ ℕ
    inv6 : nextComplianceRequest ∈ ℕ₁
    inv7 : complianceRequests ⊆ ℕ₁
    inv8 : complianceResponse ∈ ℕ
    inv9 : nextComplianceResponse ∈ ℕ₁
    inv10 : complianceResponses ⊆ ℕ₁
    inv11 : retries ∈ ℕ
VARIANT
    MAX_RETRIES − retries
```

Listing 3: Initial Model Events.

```
ReceiveMessage ≙
STATUS
    ordinary
WHEN
    grd1 : state = stWaitingForData
    grd2 : nextMessage ∈ messages
    grd3 : nextMessage ∉ receivedMessages
    grd4 : message = 0
THEN
    act1 : message := nextMessage
    act2 : receivedMessages := receivedMessages ∪{nextMessage}
    act3 : state:= stMessageToProcess
END

ReprocessMessage ≙
STATUS
    convergent
WHEN
    grd1 : state = stComplianceResponseProcessed
    grd2 : retries < MAX_RETRIES
THEN
    act1 : state := stMessageToProcess
    act2 : retries := retries + 1
END
```

4.3 Second Refinement

The second refinement focuses on getting the applicable methods for the message being processed. The second refinement is described through the UML State Diagram presented in Figure 4. Each of the substates of the GettingApplicableMethods state represents the applicability evaluation of one of the compliance method, based on their compliance scope.

The second refinement comprises four contexts and one machine. The contexts define the type ComplianceMethod, a constant ComplianceMethods (i.e. the deployed methods), a constant for each of the methods (e.g. metDQ_CountryAlpha3_GetCertificate), context

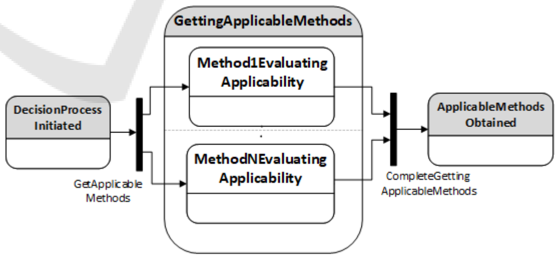
GettingApplicableMethods
Method1Evaluating Applicability
MethodNEvaluating Applicability
DecisionProcess Initiated
GetApplicable Methods
ApplicableMethods Obtained
CompleteGetting ApplicableMethods

Figure 4: 2nd Refinement - Getting Applicable Methods.

elements (e.g. ORGANIZATION, SERVICE) and functions specifying values of message properties (e.g. fromService, fromOperation), among others.

The strategy to get the applicable methods for the message being processed consists in:

1. Including two new variables in the machine (i.e. scc02) for holding the methods that have already been evaluated as applicable or not applicable.
2. For each deployed method, including two events in the machine. One should fire when the method

is applicable and the other one when it is not applicable, updating the corresponding variables.

3. Strengthening the guard of the CompleteGettingApplicableMethods event in order to continue the processing only when the applicability of all deployed methods has been evaluated.

The two new variables as well as the invariants that restrict them are presented in Listing 4.

Listing 4: Second Refinement - Machine scc02.

```
VARIABLES
    applicableMethods // current applicable methods
    notApplicableMethods // current not applicable methods
INVARIANTS
    inv1 : applicableMethods ⊆ ComplianceMethods
    inv2 : notApplicableMethods ⊆ ComplianceMethods
    inv3 : applicableMethods ∩ notApplicableMethods = ∅
```

Following this strategy, Listing 5 presents the two events for the DQ compliance method introduced in Listing 1. Note that the fourth guard of these events specifies the scope of the method using the functions for specifying message properties (e.g. fromService). Listing 5 also presents how the guard of the CompleteGettingApplicableMethods event is strengthened, in order to transition to the state stApplicableMethodsObtained only when the applicability of all the compliance methods has been evaluated.

For the second refinement of the core model, the proof obligation generator of the Rodin platform produced six invariant preservation proof obligations and one well-definedness proof obligation, which were all automatically generated and discharged.

4.4 PL4C Constructs

This section describes how the most relevant PL4C constructs are specified within the SCC Subsystem Event-B model, by leveraging Event-B elements.

Compliance methods are specified in *contexts* (by means of constants) and in different refinements (by means of events). For each compliance method, events have to be included in order to evaluate its applicability, as described in Section 4.3, and to obtain its compliance evaluation. Regarding methods applicability, two events have to be included in the model: one should fire when the method is applicable and the other one when it is not. With respect to compliance evaluation, events have to be included considering the combining algorithm of the method. Each event should fire for different conditions that leads to a compliance evaluation (e.g. *accept*, *allow*).

Message policies are specified in *contexts* (by means of constants) and in different refinements (by

Listing 5: Second Refinement Events.

```
metDQ_CountryAlpha3_GetCertificateApplicable ≙
STATUS
    ordinary
WHEN
    grd1 : decisionState = stGettingApplicableMethods
    grd2 : metDQ_CountryAlpha3_GetCertificate ∉
                    applicableMethods ∪ notApplicableMethods
    grd3 : message ∈ messages
    grd4 : (fromService(message)= DeathCertificates) ∧
                (fromOperation(message)=GetCertificate) // scope
THEN
    act1 : applicableMethods := applicableMethods ∪
                        {metDQ_CountryAlpha3_GetCertificate}
END

metDQ_CountryAlpha3_GetCertificateNotApplicable ≙
STATUS
    ordinary
WHEN
    grd1 : decisionState = stGettingApplicableMethods
    grd2 : metDQ_CountryAlpha3_GetCertificate ∉
                    applicableMethods ∪ notApplicableMethods
    grd3 : message ∈ messages
    grd4 : ¬((fromService(message)= DeathCertificates) ∧
      (fromOperation(message)=GetCertificate)) // negation of the scope
THEN
    act1 : notApplicableMethods := notApplicableMethods ∪
                        {metDQ_CountryAlpha3_GetCertificate}
END

CompleteGettingApplicableMethods ≙
    extended
STATUS
    ordinary
REFINES
    CompleteGettingApplicableMethods
WHEN
    grd1 : decisionState = stGettingApplicableMethods
    grd2 : card(ComplianceMethods) = card(applicableMethods ∪
            notApplicableMethods) // guard strengthened
THEN
    act1 : decisionState := stApplicableMethodsObtained
END
```

means of events). Similarly to methods, for each compliance policy, events have to be included in order to evaluate its applicability and to obtain its evaluation. Similarly, external response and monitored event policies are also specified in contexts and in different refinements.

Rules are also specified in contexts and in different refinements. In particular, three events have to be included for each rule in order to evaluate them: i) one that fires when the condition of the rule is true, ii) one that fires when the condition of the rule is false, and iii) one that fires when there is an error that prevents from evaluating the condition of the rule.

The complete specification of the core model as well as the model extensions for the different PL4C constructs (e.g. policies, rules) of the DQ compliance method presented in Listing 1 are available online[2].

4.5 Contributions of the Formalization

The primary contribution of the formalization of the SCC subsystem (provided by the core model) is the unambiguous specification of its general operation.

The Event-B model also enabled formal proofs of various safety properties of the core model concerning invariant preservation, convergence of events and well-definedness. This was achieved leveraging the Rodin platform which automatically generated 501 proof obligations, which were mostly either automatically or manually discharged.

The formalization also provides an unambiguous specification of how PL4C constructs (e.g. methods) extend the general operation of the subsystem. This is provided by the strategies presented in Section 4 and the extension of the core model available online.

In addition, the Event-B model enables the application of model animation and checking mechanisms in order to verify the correct operation of the SCC subsystem in specific usage scenarios (Ait-Sadoune and Ait-Ameur, 2008). Model animation complements modeling and proving by providing the means to check that the modelled system operates indeed as it was expected (Abrial, 2010). Model animation and checking was performed using ProB[3].

First, the SCC subsystem model was animated in a scenario based on the compliance method presented in Listing 1 and considering the two messages shown in Table 2: one is compliant with Alpha-3 codes (i.e. msg15) and the other one is not (i.e. msg16).

Table 2: Messages for Model Animation and Checking.

Prop. / Mess.	**msg15**	**msg16**
idMessageId	15	16
fromOrganization	MSP	MSP
fromService	DeathCertificates	DeathCertificates
fromOperation	GetCertificate	GetCertificate
msgSrvMessage	country: URY	country: URU
msgTimestamp	900	950

The results of the animation were the expected ones: both messages were delivered and an action was performed for the second message (i.e. msg16), given that it is not compliant with Alpha-3 codes.

The model checking features of ProB were also used to verify that the model has not invariant violations.

[2]https://www.fing.edu.uy/inco/grupos/lins/tesis/compliance/SCCdq.zip

[3]https://www3.hhu.de/stups/prob/

5 RELATED WORK

Several compliance management proposals leverage formal approaches for supporting their solutions. Some examples of the use of temporal logic are the Compliance Request Language (CRL) proposed by the COMPAS project (COMPAS, 2008) and the way in which compliance rules are specified within the C^{3}Pro project (Knuplesch et al., 2013). Deontic logic is used in the Business Contract Language (BLC) proposed by Governatori et al. (Governatori et al., 2006). Compared to our approach, these proposals focus on specifying compliance requirements at the business-level and they do not address the formal specification of how compliance requirements have to be controlled at the system-level within integration platforms.

Existing work also addresses the formal specification of policy-based systems. The Formal Access Control Policy Language (FACPL) is a formally-defined language for the specification, analysis and enforcement of attribute-based access control policies, inspired by XACML (Margheri et al., 2017). Other proposals use the Event-B method for formalizing some aspects of XACML policies (Errachid, 2011)(Milhau, 2011). Compared to our work, these proposals only focus on access control issues.

Finally, the Event-B method has been used in related contexts for developing formal specifications. It was used for the formal modelling of web service compositions (Ait-Sadoune and Ait-Ameur, 2015), service-oriented architecture design patterns (Tounsi et al., 2013) and BPMN models (Bryans and Wei, 2010). However, these proposals do not address the formal specification of how compliance requirements have to be controlled within integration platforms.

6 CONCLUSIONS

This paper presented a formalization of a policy-based compliance solution using the Event-B method and the Rodin platform. The solution, which is part of a broader compliance management approach, comprises a System-level Compliance Control (SCC) subsystem and a Policy Language (i.e. PL4C).

The formalization led to a core model, which specifies the general operation of the SCC subsystem, as well as an extension of this model for controlling a DQ requirement with a specific compliance method.

The main contributions of the presented formalization are the unambiguous specification of the operation of the SCC subsystem, formal proofs of safety properties of the core model and unambiguous specification of how PL4C constructs (e.g. policies) may extend the operation of the SCC subsystem. The formalization also enables model animation and checking, which provide the means to verify the correct operation of the SCC subsystem in specific scenarios.

This work also constitutes a step forward on formalizing service integration platforms and value-added services for inter-organizational environments, such as compliance control. The ultimate goal is to provide a solid ground to the specification of such platforms by assessing their correctness independently of specific implementations.

Future work includes the automatic generation of Event-B models based on compliance methods and the development of libraries containing already defined and tested compliance elements (e.g. methods, policies) for addressing specific areas of requirements (e.g. QoS). We would also analyze and propose solutions for context-aware compliance management.

REFERENCES

Abrial, J.-R. (2010). *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 1st edition.

Abrial, J.-R. (2018). On b and event-b: Principles, success and challenges. In *Lecture Notes in Computer Science*, pages 31–35. Springer International Publishing.

Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T. S., Mehta, F., and Voisin, L. (2010). Rodin: an open toolset for modelling and reasoning in event-b. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466.

Ait-Sadoune, I. and Ait-Ameur, Y. (2008). Animating event b models by formal data models. In *Communications in Computer and Information Science*, pages 37–55. Springer Berlin Heidelberg.

Ait-Sadoune, I. and Ait-Ameur, Y. (2015). *Formal Modelling and Verification of Transactional Web Service Composition: A Refinement and Proof Approach with Event-B*, pages 1–27. Springer International Publishing, Cham.

Bryans, J. W. and Wei, W. (2010). Formal analysis of bpmn models using event-b. In Kowalewski, S. and Roveri, M., editors, *Formal Methods for Industrial Critical Systems*. Springer Berlin Heidelberg.

COMPAS (2008). State of the art in the field of compliance languages. Technical report, COMPAS.

Elgammal, A., Turetken, O., van den Heuvel, W.-J., and Papazoglou, M. (2016). Formalizing and appling compliance patterns for business process compliance. *Software & Systems Modeling*, 15(1):119–146.

Errachid, M. (2011). Vérification des politiques xacml avec le langage event-b. Master Thesis.

González, L. and Ruggia, R. (2018a). A comprehensive approach to compliance management in inter-organizational service integration platforms. In *Proceedings of the 13th International Conference on Software Technologies*. SCITEPRESS.

González, L. and Ruggia, R. (2018b). On controlling compliance requirements within adaptive integration platforms. In *Proceedings of the 19th Workshop on Adaptive and Reflexive Middleware - ARM 18*. ACM Press.

González, L. and Ruggia, R. (2018c). Policy-based compliance control within inter-organizational service integration platforms. In *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE.

González, L., Ruggia, R., Abin, J., Llambías, G., Sosa, R., Rienzi, B., Bello, D., and Álvarez, F. (2012). A service-oriented integration platform to support a joined-up e-government approach: The uruguayan experience. In *Advancing Democracy, Government and Governance*, volume 7452 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Governatori, G., Milosevic, Z., and Sadiq, S. (2006). Compliance checking between business processes and business contracts. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC06)*. IEEE.

Knuplesch, D., Reichert, M., Fdhila, W., and Rinderle-Ma, S. (2013). On enabling compliance of cross-organizational business processes. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Margheri, A., Masi, M., Pugliese, R., and Tiezzi, F. (2017). A rigorous framework for specification, analysis and enforcement of access control policies. *IEEE Transactions on Software Engineering*, pages 1–1.

Milhau, J. (2011). Un processus formel d'intégration de politiques de contrôle d'accès dans les systèmes d'information. PhD Thesis.

OASIS (2013). eXtensible Access Control Markup Language (XACML) version 3.0.

Romanovsky, A. and Thomas, M., editors (2013). *Industrial Deployment of System Engineering Methods*. Springer Berlin Heidelberg.

Su, W., Abrial, J.-R., and Zhu, H. (2014). Formalizing hybrid systems with event-b and the rodin platform. *Science of Computer Programming*, 94:164–202.

Tounsi, I., Hadj Kacem, M., and Hadj Kacem, A. (2013). Building correct by construction soa design patterns: Modeling and refinement. In Drira, K., editor, *Software Architecture*. Springer Berlin Heidelberg.

Tran, H., Zdun, U., Holmes, T., Oberortner, E., Mulo, E., and Dustdar, S. (2012). Compliance in service-oriented architectures: A model-driven and view-based approach. *Information and Software Technology*, 54(6).