Constraints-based URDAD Model Verification

Fritz Solms, Priscilla Naa Dedei Hammond and Linda Marshall
Department of Computer Science, University of Pretoria, Pretoria, South Africa

Keywords: Model-driven Engineering, Model Validation, URDAD, Metamodel, Object Constraint Language, Domain-Specific Language.

Abstract: In Model-Driven Engineering the primary artifact is a technology and architecture neutral model called a Platform Independent Model (PIM). The Use-Case, Responsibility Driven Analysis and Design (URDAD) is a service-oriented method which is used to construct a PIM commonly specified in the URDAD Domain-Specific Language (DSL). In this paper we show that model quality can be verified by specifying a set of quality constraints at metamodel level which are used to verify certain consistency, completeness, traceability and simplicity qualities of URDAD models. The set of constraints has been mapped onto the Object Constraint Language (OCL) and a tool used to verify these constraints has been developed. The set of constraints is also used by an URDAD model editor to verify aspects of model quality as it is being developed.

1 INTRODUCTION

In Model-Driven Engineering (MDE) models represent the primary software engineering artifacts. They are used for communication and documentation as well artifact generation including code, test and documentation generation. Making models the primary artifact moves the emphasis of quality assurance from the code onto the models. Increasing model quality generally increases the quality of the generated artifacts and reduces project cost and risk.

(Mohagheghi and Aagedal, 2007) have shown that model quality is affected by a range of factors including (i) the quality of the modeling process used to generate the model(s), (ii) qualities of the modeling language used, (iii) the modeling tools as well as the tools used for artifact generation, (iv) the quality assurance processes and techniques applied, and (v) the knowledge and experience of the requirements analysts, architects, designers and developers.

With reference to these factors the URDAD method aims to provide (i) a quality-driven process which reduces the probability of model quality concerns (Solms et al., 2011b), (ii) a services-oriented modeling language with a metamodel specifying the language semantics and a textual as well as a diagrammatic syntax (Solms et al., 2011a), (iii) modeling tools which enable users to construct or edit a model using either a concrete text syntax or a diagrammatic syntax, and (iv) tools for model quality verification.

Model validation and verification are extensively used to assess model quality within MDE (Malgouyres and Motet, 2006; Delmas, R. et al., 2013). In model validation one establishes that the requirements model correctly reflects the stakeholder needs and the design model correctly addresses those needs. In model verification, on the other hand, one aims to establish that models are technically sound, i.e. that they are free from undesirable engineering characteristics. To reduce the risk and cost introduced by model quality concerns it is vital that model quality assessment and management is done early and continuously (Delmas, R. et al., 2013). To this end one generally aims to integrate model verification within model editing and model transformation tools.

Model verification includes the verification of the static structure of a model (e.g. for syntactic correctness, completeness, consistency and cohesion) and verification of the dynamics (processes) specified by the model in order to establish that they are free of undesirable dynamic properties like infinite loops, deadlocks and unreachable code.

In this paper we investigate the extend to which model quality can be assessed and enforced through a set of constraints specified against the URDAD metamodel. The constraints cover *consistency*, *traceability*, *completeness* and *simplicity* constraints specified using the *Object-Constraint Language* (OCL) (Object Management Group, 2012). A tool which validates URDAD models against this set of model quality con-

Solms, F., Hammond, P. and Marshall, L.
Constraints-based URDAD Model Verification.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 148-155
ISBN: 978-989-758-189-2

straints has been developed. The constraints set and tool implementation is verified through a set of test models which were constructed to contain quality defects. The OCL constraints are also used by the URDAD model editor to assess model qualities in real-time as the model is being developed. The model verification tool can be integrated into be model transformation tools in order to verify models prior to artifact generation.

2 BACKGROUND

MDE makes models the primary artifacts from which documentation, code and tests are generated. In this section we discuss relevant aspects of OMG's Model-Driven Architecture (MDA) and the URDAD analysis and design method.

The MDA (Frankel, 2003) is OMG's approach to MDE supported by a set of standards. It is based on a 4-layer layer meta-modeling architecture. The M0 layer contains the model instances (e.g. application and data instances) which are specified in the M1 layer using a model constructed using a modeling language specified in the metamodel (M2) layer which in turn is specified using a meta-language which is specified in the meta-metamodel (M3) layer.

One of MDA's main aims is to keep the requirements and application designs architecture and technology neutral such that they can be mapped onto different architectures and technologies (Frankel, 2003; Solms and Loubser, 2009). The architecture and technology neutral model is called a *Platform Independent Model* (PIM) which is incrementally transformed into a *Platform Specific Model* (PSM) and ultimately into code. The URDAD method is used to construct an URDAD model which represents a PIM.

OMG's Meta Object Facility (MOF) (Group, 2006) is a standard for specifying meta-models, i.e. a standard for specifying modeling languages. MOF is a closed meta-modeling architecture in that it provides an M3 model which conforms to itself. The UML as well as a range of other generic and domain-specific languages have been specified in MOF.

OMG has defined two compliance points for MOF namely, Essential MOF (EMOF) and Complete MOF (CMOF). The EMOF profile is a subset of MOF which was introduced to simplify the development of meta-modeling and model transformation tools. Eclipse *Ecore* is an implementation of EMOF which is supported by a suite of tools packaged within the Eclipse Modeling Framework (EMF) (Group, 2006).

Within MDA one can use either generic modeling languages like the UML or domain specific languages (Fowler, 2010) which support the semantics to model a specific domain. The latter have the advantage that they are generally much simpler, that they provide appropriate modeling abstractions and primitives closer to the ones used in the domain being modeled (Brambilla et al., 2012), have more rigorous semantics and are more amenable to code generation. The URDAD-DSL is a domain specific language for the domain of services-oriented design.

MDA includes the *Object Constraint Language* (Object Management Group, 2012), an expression language based on first order logic and set theory. The OCL is used to specify well-formedness constraints as invariant constraints on metamodels and to specify domain constraints within instance models (e.g. UML or URDAD models).

In the case of URDAD models, the OCL is used to formalize service contracts by specifying pre- and post-conditions, and to specify invariant constraints against domain objects, conditionals, variable initialization and so on. The OCL is also used as a query language to query models at any of the MOF levels (e.g. metamodels and instance models).

In this paper the OCL is used to specify a set of model quality constraints against the URDAD metamodel against which URDAD models are verified.

The *Use-Case, Responsibility Driven Analysis and Design* (URDAD) method is a service-oriented method (Solms, 2007) which was developed to make it easier for requirements specialists (e.g. business analysts) to capture and manage high-quality requirements for enterprise systems. A services oriented approach where higher-level services are orchestrated from lower level services with decoupling through services contracts is a natural modeling approach within the domain of enterprise systems development, i.e. using a services-oriented approach reduces the dichatomy between the concepts used in the problem (business) domain and the modeling domain.

The method itself is designed to encourage model qualities including simplicity, completeness, modifiability, consistency, decoupling, reusability and traceability and provides a way to manually assess these qualities (Solms et al., 2011a) in the resultant URDAD model. In this paper we study the possibility of automating the assessment of aspects of these qualities.

The URDAD model represents the PIM within OMG's MDA (Solms and Loubser, 2009), i.e. it contains the technology and architecture neutral requirements and design specification. We have used the MDA tooling provided by the Eclipse Modeling Project (Gronback and Merks, 2008) to developed the URDAD-DSL as a DSL for the domain of services

oriented design. The language semantics is specified within an *Ecore* metamodel and a concrete text syntax was defined within *EMFText* (Heidenreich et al., 2011).

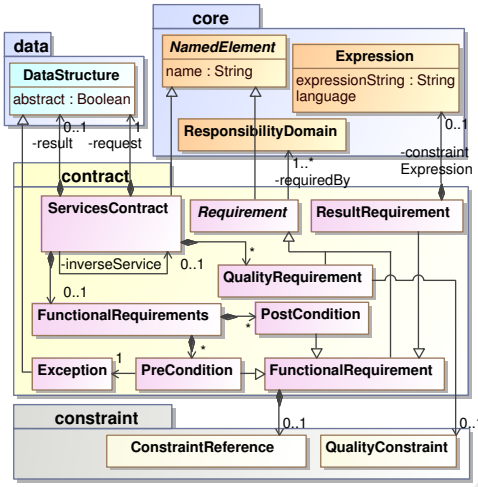

Figure 1: The contract specification elements specified in the URDAD metamodel.

The URDAD metamodel requires an URDAD model to specify services contracts representing the requirements for a service and service processes representing service designs. A service contract includes the specification of the data structures for the service request and result as well as the functional requirements. The latter are specified as pre- and post-conditions using references to reusable state constraints, i.e. the same state constraint can be a post-condition for one service and a pre-condition for another. Figure 1 depicts the contract module of the URDAD metamodel. An excerpt of an example contract specified using the concrete text syntax for the DSL is shown in Figure 2.

URDAD service specifications include the process design for the service which specifies how the service is orchestrated from lower level services realizing lower level services contracts, i.e. services across levels of granularity are decoupled through the service contract specifications[1]. Conceptually each service specification represents a template method specification for which the concrete realizations of the process steps can vary. To facilitate traceability the service specification is associated with a services contract and includes the link between lower level services used

[1]When mapping the technology neutral design specification onto code the decoupling between services across different levels of granularity is often retained by making use of dependency injection frameworks which provide handles to concrete services which are to be used to realize service contracts at run-time.

```
ServiceContract enrollForPresentation
{
  FunctionalRequirements receiving Variable enrollForPresentationRequest ofType
          EnrollForPresentationRequest
  {
    PreCondition enrollmentPrerequisitesMet
        requiredBy (TrainingRegulator Student)
        raises EnrollmentPrerequisitesNotSatisfiedException
        checks Constraint enrollmentPrerequisitesForPresentationMet
        with valueOf enrollForPresentationRequest
    ...
    PostCondition enrollmentProcessPerformed
        requiredBy (Student Client TrainingRegulator)
        ensures Constraint studentEnrolledForPresentation
        with valueOf studentEnrolledRequest constructedUsing doSequential
        {
          create Variable studentEnrolledRequest ofType StudentEnrolledRequest
          set Query OCL:"studentEnrolledRequest.personIdentifier" equalTo
              Query OCL:"enrollForPresentationRequest.personIdentifier"
          ... } }
    Request DataStructure EnrollForPresentationRequest
    {
        has Identification presentationIdentifier identifying Presentation
        ... }
    Result DataStructure EnrollForPresentationResult
    {
        has Component proofOfEnrollment ofType ProofOfEnrollment
        ... } } }
```

Figure 2: An excerpt of the contract specification for an enrollment service.

and the functional requirements for which they are required. The latter is done through the `use ... toAddress` relationships. An excerpt of an example service specification using the concrete text syntax for the DSL is shown in Figure 2.

Even though the URDAD method specifies a "quality-driven analysis and design process" which guides analysts to construct a requirements and technology and architecture neutral design model exhibiting certain model qualities, the method itself does not enforce these qualities (Solms et al., 2011b). When considering model quality it is important to realize that model quality is affected by the quality characteristics of artifacts from different levels of abstraction within the MDA metamodeling architecture as well as by the concrete syntax used by the modeling language. In particular, *semantic quality*, *degree of formalization*, *conceptual simplicity*, *traceability*, *language consistency* are characteristics which are determined within the M2-layer, i.e. the quality of the metamodel specifying the modeling language. On the other hand, *understandability* and *ease of use* are qualities which are determined by the concrete syntax used to specify the model (Solms et al., 2011a). Finally, *syntactic correctness consistency*, *completeness*, *correctness*, *simplicity* including *lack of redundancies*, *uniformity*, and *cohesion* apply to the M1 layer, i.e. they are qualities of specific URDAD models. Of these model qualities *correctness* falls within the scope of model validation whilst the other qualities fall within the scope of model verification.

The metamodel and the language aware model editor we have developed enforce *syntactic correctness* (that an URDAD model complies to the struc-

```
Service enrollForPresentation realizes enrollForPresentation
  receiving Variable enrollForPresentationRequest ofType
            EnrollForPresentationRequest
{
  use checkStudentSatisfiesEnrollmentPrerequisites toAddress (
            enrollmentPrerequisitesMet)
  use issueInvoice toAddress (financialPrerequisitesSatisfied invoiceIssued)
  ...
  Process doSequential
  {
    create Variable checkStudentSatisfiesEnrollmentPrerequisitesRequest
       ofType CheckStudentSatisfiesEnrollmentPrerequisitesRequest
    set Query OCL:"checkEnrollmentPrerequisitesRequest.studentIdentifier"
      equalTo Query OCL:"enrollForPresentationRequest.studentIdentifier"
    ...
    requestService checkStudentSatisfiesEnrollmentPrerequisites
       with checkStudentSatisfiesEnrollmentPrerequisitesRequest
       yielding Variable checkStudentSatisfiesEnrollmentPrerequisitesResult
         ofType CheckStudentSatisfiesEnrollmentPrerequisitesResult
    choice
    {
       if Constraint enrollmentMeetsPrerequisitesMet
         OCL:"if checkStudentSatisfiesEnrollmentPrerequisitesResult.isOclKind(
                StudentSatisfiesEnrollmentPrerequisites)"
         doSequential
         {
            create Variable issueInvoiceRequest ofType IssueInvoiceRequest
            set Query OCL:"issueInvoiceRequest.clientIdentifier" equalTo
            Query OCL:"enrollForPresentationRequest.clientIdentifier"
            ...
            requestService issueInvoice with issueInvoiceRequest
               yielding Variable issueInvoiceResult ofType IssueInvoiceResult
            {
               on FinancialPrerequisitesNotSatisfiedException
                  raiseException FinancialPrerequisitesNotSatisfiedException
            }
            ...
            returnResult enrollForPresentationResult
       }
       else raiseException EnrollmentPrerequisitesNotSatisfiedException
  } } }
```

Figure 3: An excerpt of a service (process) specification realizing a services contracts.

ture specified by the metamodel) as well as *uniformity* (that the same model features are consistently used to specify similar model elements). Note that unlike the UML, the URDAD-DSL is not overloaded. For example, whilst in the UML processes can be specified using sequence diagrams, activity diagrams, state charts, interaction-overview diagrams, communication diagrams or timing diagrams, the URDAD-DSL only provides one model feature to specify the process for a service.

3 MODEL CONSTRAINTS

The URDAD metamodel specifies the semantics available to specify an URDAD model as well as the structure of such a model. However, compliance of an URDAD model to the metamodel is not sufficient to guarantee model quality.

For example, URDAD being a contracts-based services oriented approach which facilitates traceability across functional requirements specified within a service contact and the service (process) specification through which the service is realized requires models to specify (a) that a service declares the lower level services through which pre- and post-conditions of the contract are addressed, (b) that services which were declared to be used to address the functional requirements are called in the process specification, (c) that a service raises only exceptions which are associated with a pre-conditions specified for the service contract the service realizes, and (d) that any exceptions raised by lower level services are either handled or are associated with pre-conditions of the calling service.

These as well as a range of other structural constraints are mapped onto a formal representation as OCL invariance constraints (Object Management Group, 2012). This enables us to have a stand-alone model verification tool using the OCL tool support built into the Eclipse Modeling Framework (Gronback and Merks, 2008) to verify instance models prior to model transformation (e.g. in the code generation, unit test generation and documentation generation). In addition we have integrated the constraint suite within the URDAD model editor to provide real-time model verification to requirements analysts and process designers.

The structural constraints introduced to assess aspects of URDAD model quality can be grouped into completeness, consistency and simplicity constraints.

3.1 Completeness Constraints

This section discusses some constraints whose violation indicates that the quality of an URDAD model might be insufficient because the structure of the model is incomplete.

The first set of completeness constraints enforces the traceability of how functional requirements are addressed, i.e. that one can trace the realization of each pre- and post-condition to the lower level services used to address them. Taing into account the structure of the URDAD metamodel, this constrained is factored into two lower level constraints:

1. For each functional requirement (i.e. pre- or post-condition) the service contract specifying the requirements for lower level services required to address the functional requirement is specified. In the concrete text syntax this is done via a `use xxx toAddress` construct which feeds the appropriate association between the functional requirement and the required service into the URDAD metamodel:

```
context Service
inv AllConditionsAddressed: serviceRequirements->isEmpty() or (
        serviceRequirements.usedToAddress->includesAll(realizedContract.
        functionalRequirements.preConditions) and serviceRequirements.
        usedToAddress->includesAll(realizedContract.functionalRequirements.
        postConditions));
```

2. All `toAddress` functions must be called in the process specification

```
context Service
inv AlltoAddressFunctionsInProcessSpecification: serviceRequirements->
        isEmpty() or serviceRequirements.requiredService->includes(process.
        oclAsType(ActivitySequence).activities->flatten()->collect(
        RequestService.allInstances().requestedService))
```

Note the enforced decoupling within the URDAD model. The dependencies in the process specification are purely on service contracts for lower level services and not on concrete services which realize the service contracts specifying the service requirements. Within concrete implementations these service contracts are bound to implementing services either at compile, deploy or run-time. Run-time binding is commonly done through either run-time services lookups within service registries or using dependency injection frameworks.

Not all completeness constraints are necessarily indicative of model errors. For example, the constraint that all service contracts have service implementations is used to project out those service contracts for which no service implementations have been defined. However, since services can be sourced from outside the system (e.g. standard libraries, frameworks, ...) and since such services could be injected at run-time, a violation of this constraint does not necessarily imply a model completeness defect. The corresponding OCL constraint listed below is thus contained in a separate set of OCL constraints which is used to generate warnings upon constraint violation. Service contracts for which no service specifications are contained in the model thus flag services which need to be sourced front the environment.

```
context ResponsibilityDomain
inv ServiceContractHasAServiceSpecification: servicesContracts->collect(sc :
        contract::ServiceContract | services.realizedContract->excludes(sc))->size()
        > 0
```

3.2 Consistency Constraints

The design of the URDAD meta-model was done as to have much lower language complexity and inconsistency risks than a generic language like the UML (Solms et al., 2011a). Nevertheless, meta-model compliance alone does not enforce model consistency.

To be able to verify model consistency a range of consistency constraints is included in the set of OCL-based model constraints. Many of these constraints are very simple constraints like constraining the minimum of a range to be less than or equal to the maximum, constraining service request parameters in service process specifications to be an instance of the service the request type of the service contract and so on.

For illustrative purposes we choose one of the consistency constraints and discuss it in some detail. URDAD is a contracts-based approach which differentiates between errors and exceptions. In particular, an exception is raised to communicate to the client that the requested service is not being provided because a particular pre-condition for that service was not met. For example, the debit service of an account may have the pre-condition that the available balance must be equal to or greater than the withdrawel amount. Should this pre-condition not hold, the service is refused and the client is informed of the service refusal by raising that exception which has been associated with this precondition. Note that the non-provision of the service is, in this case, not caused by a system error. An error, on the other hand, is the admission that the service is unable to fulfill its contractual obligations. The structural requirement that one has to associate with each pre-condition an exception is enforced by the meta-model itself, i.e. the model needs to be only verified against the meta-model and no further constraints are required.

On the other hand, the requirement that a process specification for a service does not raise any exceptions other than those associated with pre-conditions of the service as specified in the service contract is not enforced structurally, i.e. by the meta-model. We need to thus include in the model quality constraints set a constraint which can be used to verify that any process specification for a service raises only exceptions which are associated in the service contract with one of the pre-conditions of the service. The following OCL constraint requires that a process raises only exception objects which are instances of either one of the exception classes or one their sub-classes.

```
context Service
  inv AllExceptionsInProcessAreDeclaredInContract: contract::Exception.allInstances()
        ->includes(process.oclAsType(ActivitySequence).activities->collect(act :
        Activity | act.oclAsType(ExceptionHandler).exception)) and
                                        (contract::Exception.allInstances()->
                                                includes(process.oclAsType(
                                                ActivitySequence).activities->
                                                collect(act : Activity | act.
                                                oclAsType(RaiseException).
                                                exception)))
```

3.3 Simplicity Constraints

Unnecessary complexity is viewed as a model quality concern. It results in reduced understandability, increased maintenance costs and risk and potentially in performance overheads. A thorough complexity analysis is beyond the scope of a constraint set specified against the metamodel.

On the other hand, the constraint that a process should only request lower level services which have been identified to be used to contribute to addressing the functional requirements for the service as specified in the service contract can be specified as follows:

```
context Service
  inv AllLowerLevelServicesUsedMustAddressFunctionalRequirements:
        serviceRequirements->forAll(serReq : ServiceRequirement | serReq.
```

```
        usedToAddress->notEmpty())
```

The set of simplicity constraints also includes more standard aspects like the constraint that a process specification should not include any variables which are not used.

```
context Service
  inv EveryCreatedVariableMustBeUsed: process.oclAsType(ActivitySequence).
        activities->collect(ManipulateVariable.allInstances().source)->collect(mv
        : ManipulateVariable | mv.source.name.substring(1, mv.source.name.indexOf
        ('.')-1))->includesAll(process.oclAsType(ActivitySequence).activities->
        collect(VariableProduction.allInstances().producedVariable.name))
```

4 MODEL VERIFICATION TOOL

A model verification tool is a simple Java application developed using the Eclipse Modeling Framework (EMF) (Steinberg et al., 2009). In particular, the tool uses the EMF OCL framework for OCL constraints parsing. The tool takes as inputs the URDAD metamodel as an Ecore file encoded in XMI, the OCL constraints file and the XMI file for an URDAD instance model and reports a list of quality constraint violations. In addition, the OCL constraint suite is used by our text-based URDAD-DSL editor to validate models in real-time and to highlight model quality concerns as models are being developed.

5 RELATED WORK

(Lindland et al., 1994) identified three types of model quality: (i) *Syntactic quality* as the degree to which a model adheres to the rules of the modeling language. (ii) *Semantic quality* representing the accuracy with which the problem domain is modeled. (iii) *Pragmatic model quality* representing the degree to which the model can be pragmatically used – e.g. for communication as well as code, test and documentation generation.

Semantic quality includes the notion of *feasible functional completeness* which is the degree to which all relevant functional requirements are included. Within an URDAD model this is the degree to which (a) contracts for lower level services used to address pre- and post-conditions have been specified, (b) for each service contract for which the service is not sourced externally, there is a service specification in the model.

Fieber, Huhn and Rumpe (Fieber et al., 2008) provide a taxonomy of quality characteristics for models. They differentiate between *inner model qualities* which can be assessed by considering the model in isolation and *outer model qualities* which are assessed relative to the model context. Inner model qualities include the semantic quality, the degree of formalization, understandability, simplicity, conceptual integrity/uniformity and conformity to standards and norms. Outer model qualities include completeness, lack of redundancies, cohesion which is related to modularization, sufficiency for application.

Within MDA-based approach the *Inner model qualities* can be mapped onto qualities of the metamodel which specifies the modeling language used to specify instance models and outer model qualities can be mapped at qualities against which instance models are assessed. Aspects of the qualities of an URDAD metamodel have been analyzed in (Solms et al., 2011a). The focus of this paper is to be able to verify aspects of *outer model qualities*, i.e. qualities of instance models.

(Malgouyres and Motet, 2006) enriched the UML with a set of consistency constraints. Given an instance of a UML model, they map both, the model and the consistency constraints onto a logic statements specified in a *Constrained Logic Programming* (CLP) language (Cohen, 1996) and used a CLP solver to assess the internal consistency of the statements derived from the UML model as well as the adherence of the model to the consistency constraints specified against the UML metamodel.

Since (Malgouyres and Motet, 2006) focus on consistency checking of UML models, the set of consistency constraints is necessarily limited to some technical consistency checks which apply to any UML model. Since we are specifying constraints against URDAD models which have a more constraint yet more rigorously defined semantics our constraints can be used to assess completeness (including traceability), simplicity and consistency qualities which are specific to services-oriented URDAD models. However, our approach can be augmented to a mapping of URDAD instance models onto CLP in order to more rigorously verify the internal consistency of the semantic statements contained in the model.

CLP has also been used to assess the consistency of domain models represented by UML class diagrams and associated M1-level OCL invariance constraints (Cabot et al., 2008). The authors map the semantics specified by class diagrams refined with invariance constraints onto a *Constraint Satisfaction Problem* (*CSP*) and used the ECL^iP^e constraint library (Aggoun, 2006) to verify internal model consistency by assessing the instantiability of the CSP. In contrast, we specify OCL invariance constraints assessing certain model qualities against the M2-level meta-model. The purpose of the two approaches is quite different with our approach being able to assess

model qualities like traceability, the degree to which pre- and post-conditions are being addressed and so on, whilst the work of (Cabot et al., 2008) focuses on internal consistency of the model including any M1-level constraints specified against the model.

(Lange and Chaudron, 2005) considered different model uses in order to develop a quality model for UML models as well as a model quality visualization tool. Of the model qualities included in their quality model, traceability, consistency and completeness are included within our model quality verification tool. Others like aesthetics and conciseness are qualities of a concrete syntax and are not the topic of this paper. Correspondence (that model qualities directly correspond with system elements) is not a desirable for an URDAD model as it represents a technology and architecture neutral application model.

(Berkenkötter, 2006) developed a UML profile containing railway-specific concepts and constraints. The constraints were specified in the OCL and were validated by the *Eclipse OCL Engine*. However, due to the complexity and weak sematics of the UML, designers are typically required to use a specific subset of the language in a specific way and a large and complex set of constraints is required to validate model quality. The URDAD DSL which was developed to reduce language complexity and improve language semantics requires comparatively a much smaller set of model quality constraints.

Mappings onto formal process specification languages have been performed for UML sequence diagrams (Dan, 2010), activity diagrams (Elmansouri et al., 2011) and state charts (Ng and Butler, 2003; Delmas, R. et al., 2013). The formal process specifications are then verified by a model checker to confirm the absence of a set of process design flaws. Model verification has also been applied to OWL-S (Zhi-Jun et al., 2005) used for semantic markup for web service composition (Xia and Li, 2009).

URDAD is itself a service-oriented method which is used to specify service contracts and process specifications across levels of granularity. We are busy mapping URDAD process specifications onto a formal specification language for formal process specification, but this is not the topic of this paper.

6 CONCLUSIONS AND FUTURE WORK

Within model-driven engineering models are the primary artifacts and model quality is critical. It is determined by the semantic quality of the modeling language, qualities of the concrete syntax used and qualities of the actual instance models. The latter includes syntactic quality which can be assessed by verifying conformance to the meta-model (language conformance), qualities of the model structure and qualities of the dynamics (i.e. the processes) specified in the model.

In this paper we investigated the extend to which we were able to specify completeness (including traceability), consistency and simplicity constraints against the URDAD meta-model in order to verify these qualities within URDAD models. The resultant set of OCL constraints was tested using a set of test models. A model verification tool enables us to verify URDAD model quality. The OCL constraints are also used by the URDAD model editor to verify models as they are being developed.

Whilst our approach does verify aspects of model consistency, the consistency of constraints specified against the instance model are not currently verified. Future work will look at mapping URDAD instance models onto a Constraint Satisfaction Problem (Cabot et al., 2008) in order to increase the level of consistency checking. Also, the quality of processes specified within URDAD models is currently not verified.

REFERENCES

Aggoun, A. e. a. (2006). The *ecl^{i}p^e* user manual. http://eclipseclp.org/doc/userman/umsroot.html.

Berkenkötter, K. (2006). Ocl-based validation of a railway domain profile. In *Proceedings of the 2006 International Conference on Models in Software Engineering*, MoDELS'06, pages 159–168, Berlin, Heidelberg. Springer-Verlag.

Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 1st edition.

Cabot, J., Clariso, R., and Riera, D. (2008). Verification of uml/ocl class diagrams using constraint programming. In *Software Testing Verification and Validation Workshop, 2008. ICSTW '08. IEEE International Conference on*, pages 73–80.

Cohen, J. (1996). Logic programming and constraint logic programming. *ACM Comput. Surv.*, 28(1):257–259.

Dan, L. (2010). QVT Based Model Transformation from Sequence Diagram to CSP. In *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*, pages 349–354.

Delmas, R., Pires, A. F., and Polacsek, T. (2013). A verification and validation process for model-driven engineering. In Array, editor, *EUCASS Proceedings Series Advances in AeroSpace Sciences*, volume 6, pages 455–468.

Elmansouri, R., Hamrouche, H., and Chaoui, A. (2011). From UML Activity Diagrams to CSP Expressions: A Graph Transformation Approach using Atom3 Tool.

International Journal of Computer Science Issues, 8(2):368–374.

Fieber, F., Huhn, M., and Rumpe, B. (2008). Modellqualität als indikator für softwarequalität: eine taxonomie. *Informatik-Spektrum*, 31(5):408 424.

Fowler, M. (2010). *Domain Specific Languages*. Addison-Wesley Professional, 1st edition.

Frankel, D. S. (2003). *Model Driven Architecture: Applying MDA to enterprise computing*. John Wiley & Sons, New York.

Gronback, R. C. and Merks, E. (2008). Model Driven Architecture at Eclipse. *The European Journal for the Informatics Professional*, 2008(II).

Group, O. M. (2006). *Meta Object Facility (MOF) Core Specification Version 2.0*.

Heidenreich, F., Johannes, J., Karol, S., Seifert, M., and Wende, C. (2011). Model-based language engineering with emftext. In Lmmel, R., Saraiva, J., and Visser, J., editors, *GTTSE*, volume 7680 of *Lecture Notes in Computer Science*, pages 322–345. Springer.

Lange, C. and Chaudron, M. (2005). Managing Model Quality in UML-Based Software Development. In *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on*, pages 7–16.

Lindland, O., Sindre, G., and Solvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software*, 11(2):42–49.

Malgouyres, H. and Motet, G. (2006). A uml model consistency verification approach based on meta-modeling formalization. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 1804–1809, New York, NY, USA. ACM.

Mohagheghi, P. and Aagedal, J. (2007). Evaluating Quality in Model-Driven Engineering. In *MISE '07: Proceedings of the International Workshop on Modeling in Software Engineering*, pages 6–11, Washington, DC, USA. IEEE Computer Society.

Ng, M. Y. and Butler, M. (2003). Towards formalizing UML state diagrams in CSP. In *Software Engineering and Formal Methods, 2003.Proceedings. First International Conference on*, pages 138–147.

Object Management Group (2012). OCL 2.3.1 Specification.

Solms, F. (2007). Technology Neutral Business Process Design using URDAD. In *Proceeding of the 2007 conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the sixth SoMeT-07*, pages 52–70, Amsterdam, The Netherlands, The Netherlands. IOS Press. ACM ID: 1566976.

Solms, F., Edwards, C., Paar, A., and Gruner, S. (2011a). A Domain-Specific Language for URDAD Based Requirements Elicitation. In *Proceedings of the South African Institute of Computer Scientists and Information Technologists Conference on Knowledge, Innovation and Leadership in a Diverse, Multidisciplinary Environment*, SAICSIT '11, pages 224–230, New York, NY, USA. ACM.

Solms, F., Gruner, S., and Edwards, C. (2011b). URDAD as a Quality Driven Analysis and Design Process. In *New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Tenth SoMeT 11*, volume 231, pages 141–158. IOS Press BV. ACM ID: 1566976.

Solms, F. and Loubser, D. (2009). Generating MDA's platform independent model using URDAD. *Knowledge-Based Systems*, 22(3):174–185.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

Xia, H. and Li, Z. (2009). Verification web services composition based on owl-s. In *Knowledge Acquisition and Modeling, 2009. KAM '09. Second International Symposium on*, volume 1, pages 164–167.

Zhi-Jun, D., Jun-li, W., and Chang-Jun, J. (2005). Semantic web service composition based on owl-s. In *Semantics, Knowledge and Grid, 2005. SKG '05. First International Conference on*, pages 98–98.