A Domain-Specific Language for URDAD Based
Requirements Elicitation

F. Solms\textsuperscript{1}, C. Edwards\textsuperscript{1}, S. Gruner\textsuperscript{1}, A. Paar\textsuperscript{1}, D. Loubser\textsuperscript{2}

\textsuperscript{1} Dept. of Comp. Sc., Univ. of Pretoria, South Africa, \texttt{fritz@solms.co.za}
\textsuperscript{2} Solms Training and Consulting CC, 2109 Johannesburg, South Africa

\textbf{Abstract.} Use-Case Responsibility-Driven Analysis and Design (URDAD) is a services-oriented software analysis and design methodology. It is used by requirements engineers to develop technology-neutral, semi-formal platform-independent models (PIM) within the OMG’s MDA. In the past, URDAD models were denoted in UML. However, that was a tedious and error-prone. The resulting models were often of rather poor quality. In this paper we introduce and discuss a new Domain-Specific Language (DSL) for URDAD. Its meta model is consistent and satisfiable. We show that URDAD DSL specifications are simpler and allow for more complete services contract specification than their corresponding UML expressions. They also enable traceability and test case generation.

\textbf{Keywords:} model-driven development, domain-specific language, meta model, service-orientation, requirements engineering, platform-independent model

1\ Introduction

Insufficiency in requirements engineering is still regarded as a root cause of poor software quality. This is due to various factors, both human and technological, including vague specification languages with only informally defined semantics. Insufficient language support for \textit{layered} specifications (i.e., decompositional system descriptions at different levels of granularity), leads software developers to making wrong presumptions about lower level requirements \cite{5}. Tool support for the validation of requirements specifications, or for the automatic extraction of test cases from them, is also still weak \cite{2}.

Model-Driven Engineering (MDE) \cite{17} aims at solving some of those problems by using modelling languages with well defined semantics, by requiring primary models to be domain models, not technical models \cite{1} and by providing tool support for MDE processes. Consequently, technology-neutral domain models are developed by requirements specialists, not by technical experts \cite{1}.

URDAD, the Use-Case Responsibility-Driven Analysis and Design methodology \cite{18} supports MDE in a service-oriented way \cite{19}. It is used by requirements specialists to develop and validate technology-neutral requirements models. URDAD models are thus Platform-Independent Models (PIM) in the Model-Driven Architecture (MDA) context \cite{19}. For each level of granularity the method leads
to testable service contracts and for non-leaf services a technology neutral process realizing the service contract through the use of lower level services. Higher-level services are thus a functional composition of lower-level services, similar to the classical DFD technique [20], with the levels of granularity decoupled through services contracts.

In the past, requirements engineers had used the Unified Model Language (UML) to encode URDAD models. UML was a reasonable choice for this purpose because of its tool-supported use in the software industry. In that approach however, the responsibility to only use an URDAD-appropriate subset of the UML (and thus to comply with the intended URDAD modelling rules) was entirely left to requirements engineers. Indeed, a specific UML profile could be used to restrict the use of UML according to URDAD’s intentions. In practice, however, the UML approach to URDAD turned out to be so tedious and error-prone that subsequent model transformations were often not feasible.

In this paper we present a new domain-specific language (DSL) for the domain of technology-neutral, services-oriented requirements modelling. Our new URDAD DSL is described in terms of a MOF/EMOF meta model. This makes it amenable to MDA tool suites for model transformations, as well as the generation of concrete textual and diagrammatic syntaxes with tool support [7]. To this end we analyse theoretically the modelling constructs required by URDAD. We elucidate and critically assess the URDAD meta model, and we propose a concrete textual syntax for an URDAD DSL. A Description Logics (DL)-based representation of the URDAD meta model is derived from the MOF/EMOF meta model in order to show its consistency and satisfiability.

Consequently we argue (also w.r.t. related work) that the URDAD DSL has two main advantages over the use of an URDAD UML profile. The language is considerably simpler than the UML and, with appropriate tool support, is expected to simplify the process through which requirements engineers can build high-level, technology-neutral models. Our new DSL enforces the structure required for a valid URDAD model, thereby requiring only a rather small and simple set of meta model constraints at the basis of tool-supported model validation. In addition the URDAD DSL provides better support for specifying services contracts within a services oriented approach.

2 Overview of the URDAD Methodology

URDAD is currently a semi-formal methodology for requirements elicitation in the form of layered services contracts and processes [19]; its formalisation is the topic of ongoing research. Business processes are specified as services with corresponding service contracts and process flows. The URDAD methodology stipulates a repeatable engineering workflow on the basis of the following iterated steps: service contract specification, responsibility allocation, and process design. The methodology envisages requirements specialists across responsibility domains to contribute to a single requirements model.
Service Contract Specification. The URDAD methodology facilitates the incremental refinement of service requirements across different levels of granularity. Service requirements are encapsulated within a service contract. The specification of a service contract includes the identification of stake holders, the functional and quality requirements, and the data structures required for the service’s request and result objects. A stake holder may be a role, or another service.

Functional service requirements can be expressed in terms of pre- or post-conditions. If all preconditions of a service are met, the service must be provided. The specification of each pre-condition includes an exception type which must be raised to notify the service requester that the requested service is refused due to the associated pre-condition not being met. A specified post-condition must hold true after the service has been provided. Non-functional service requirements stipulate qualities such as scalability, efficiency, reliability, accessibility, security, etc.

Responsibility Allocation. During the responsibility allocation step the lower level services that are used to assemble a higher level service are identified by their ability to address the higher level service’s functional requirements. Many of URDAD’s concepts originate from Responsibility Driven Design (RDD) [22, 21]. Each service contract and its corresponding service are assigned to a responsibility domain. This prompts domain experts to search within an appropriate responsibility domain for existing services that can be re-used to implement functional requirements.

Service contracts represent requirements on a specific level of granularity. The complete requirements for a service are accrued by the accumulation of its requirements and those of its required lower level services. The hierarchical decomposition of service contracts has made requirements engineering better manageable. Additional levels of granularity can be opened by coalescing several service requirements into a single cohesive super-service. Such abstraction from details (principle of information hiding) reduces the intellectual complexity, improves the understandability of requirements elicitation tasks and opens further reuse opportunities.

Process Design. Computational processes are specified using services identified in the responsibility allocation step, i.e. processes are ‘orchestrated’ across lower level services used to realize the pre- and post-conditions of the service. A process specification is assembled from standard control logic for sequential, concurrent and conditional activities with activities either constructing and request objects or the object representing the computational output, requesting lower level services, handling exceptions raised by lower level services or raising an exception or returning a result to the user of the service. Each path through a process graph must either end with an output, or with raising an exception associated with a precondition of the service.
3  URDAD’s Meta Model

URDAD’s meta model provides a formal description of URDAD’s domain of discourse. It will be used to support the URDAD methodology of requirements specification. A meta model is a “logical information model that specifies the modeling elements used within another (or the same) modeling notation” [12].

URDAD’s meta model formalizes the modelling constructs needed for the specification of service requirements and the technology-neutral design of business processes which fulfil such requirements. It is specified in Ecore which is an implementation of EMOF. The Eclipse modelling tool suite[7] supports the automatic generation of concrete textual and diagrammatic grammars, QVT-based model transformations, as well as the integrated use of the Object Constraint Language (OCL) [8]. An automatic translation of the meta model into a representation within the Web Ontology Language (OWL DL) ontology was done to facilitate automated satisfiability checking (see Section 4).

Our meta model was designed to contain the smallest sufficient set of concepts that describe URDAD’s service-oriented analysis and design specifications. It was influenced by the UML and the Business Process Execution Language (BPEL) in the area of data structure and process specification. One of its underlying assumptions is the internal ‘statelessness’ of services; ‘state’ is assumed to be stored in the services’ environments. The metamodel describes the notions of a responsibility domain, intra-model relationships facilitating traceability, service contracts including pre-conditions and their associated exceptions, post-conditions, as well as service-specific request and result classes. Our DSL also describes a system of parametrised, reusable operational constraints, which are applied to information obtained from the environment sourced through services.

Using tools like EMFText [9] one can either generate or specify a textual grammar for a metamodel, and from this grammar both a parser and a language-aware editor supporting syntax checking and auto-completion. In this section we illustrate core aspects of the URDAD DSL through an example encoded in the grammar which we have defined for the language. The textual grammar is meant to enable requirements specialists to specify URDAD DSL models within a simple and intuitive textual syntax. Ultimately we expect a graphical syntax to be still more accessible for requirements specialists than the textual syntax.

3.1  The Core of URDAD’s Meta Model

The core module (see Figure 1) introduces core concepts like the URDAD model itself, model elements, stake holders, responsibility domains, expressions and annotations.

A responsibility domain covers a coherent unit of functionality at a particular level of granularity. Responsibility domains are meaningful groupings of service contracts. Thereby our notion of ‘responsibility domain’ is similar to the notion of ‘unity criteria’ in [6]. Technically, responsibility domains are packaging constructs to subsume model elements under a unique name space. Note that they also define the boundaries between different levels of granularity.
3.2 Constraints

Constraints are required for the specification of functional requirements (pre- and post-conditions), data structure constraints and decision points in processes. The Object-Constraint Language (OCL) has become the de-facto standard for specifying constraints across object graphs. However, in a services-oriented approach and in the context of reusable, parametrised constraints the OCL alone is not expressive enough.

Firstly, in a service-oriented context the actual environmental state is only accessible by using services which query the environment and not by traversing an object graph. The specification of constraints must thus relate to the specification of services through which information is sourced from the environment together with a set of data structure constraints on the obtained information. OCL can be used for the latter, but is insufficient for the complete specification of a constraint.

Secondly, the definition of reusable constraints requires support for binding parameters. For example, assume a constraint that some ‘Person’ must be ‘registered’. Such a constraint can contribute to pre- or post-conditions of services. Hence, to be able to do so, the person identifier would have to be passed as a parameter to the constraint entity.

The following listing shows an extract of the textual representation of our ‘registration’ example. It illustrates the specification of a simple parametrised constraint which includes the specification of a process that extracts information from the environment as well as a data constraint to be checked against the such-obtained information.

```plaintext
StateConstraint studentEnrolledForPresentation receiving Variable enrollForPresentationRequest ofType EnrollForPresentationRequest
{
    stateAssessmentProcess doSequential
    {
        create Variable getEnrollmentsRequest ofType GetEnrollmentsRequest
        set Query OCL:"getEnrollmentsRequest.presentationIdentifier" equals Query OCL:"enrollForPresentationRequest.presentationIdentifier"
        requestService getEnrollments with getEnrollmentsRequest yielding Variable getEnrollmentsResult ofType GetEnrollmentsResult
    }
}
```

Fig. 1. The core elements of URDAD
As depicted in Figure 2, the constraints module provides the concept of re-usable constraints with standard logical operators to formulate complex constraint expressions.

### 3.3 Service Contracts

A service contract comprises both functional and non-functional quality requirements. Every requirement is associated with a stakeholder which is either a responsibility domain or another service. Functional service requirements are expressed in terms of the above-mentioned pre- and post-conditions, together with exception rules for the case that any pre-condition is not fulfilled.

A post-condition can specify either the computational output of a service, or its side effects on its environment. Due to a service potentially creating a lasting change to its environment, the URDAD DSL allows the designation of inverse services through which these lasting effects can be reversed. Explicit post-conditions thus also help to specify such undo-services.

A service contract specifies a single service request object that contains information pertaining to the request and a single result object, which contains the information associated with the result of the executed service. The data structures for these request and result objects are service-specific and are not meant to be reused (though their components, which are domain objects, are most likely to be reused).

Below we illustrate the specification of a service contract in our URDAD DSL grammar.
Listing 1.2. Specifying a service contract in the textual URDAD DSL syntax.

Figure 3 shows the URDAD DSL support for the specification of services contracts as packaged within a contract module.

### 3.4 Data Structures

The data module of the URDAD meta model is depicted in Figure 4. It allows for an object-oriented approach to data structure specification which is directly aligned with UML class descriptions. In addition to the familiar type relationships - Association, Aggregation and Composition, URDAD DSL introduces a new, weaker type relationship called Identification. This is conceptually similar to a strongly-typed Uniform Resource Identifier (URI), in that it uniquely identifies objects within the model, but without implying an active message path to the object as UML Associations do.
3.5 Processes

The process package of the URDAD meta model describes the concept of a service as a concrete unit of functionality in fulfillment of its service contract. The responsibility allocation step of the URDAD methodology is supported by capturing the lower level service requirements for a service, relating each service used to either the assessment of a pre-condition or the realization of a post-condition. This is done via `usedToAddress` links representing satisfaction links as in [16]).

The second aspect of a service is the specification of a process which is assembled from service requests associated with the lower level services contracts used to address the functional requirements of the service. Note that decoupling of levels of granularity via services contracts is enforced. The URDAD metamodel assumes that the selection of concrete service providers for the lower level services is either done by the deployment environment through mechanisms like dependency injection or specified during the implementation mapping phase.

The URDAD metamodel also contains explicit modelling constructs for creating and manipulating local process variables, for handling an exception raised by a lower level service, for raising an exception associated with a pre-condition of the service, and for the return of a computational result. The following listing illustrates how a service contract is denoted in the grammar of our URDAD DSL.

```plaintext
Service enrollForPresentationImpl realizes enrollForPresentation
  receiving Variable enrollForPresentationRequest ofType
  EnrollForPresentationRequest
```
\{ 
use checkStudentSatisfiesEnrollmentPrerequisites toAddress (enrollmentPrerequisitesMet)
use issueInvoice toAddress (financialPrerequisitesSatisfied invoiceIssued)
use performEnrollment toAddress (invoiceIssued)

Process doSequential 
{
create Variable checkStudentSatisfiesEnrollmentPrerequisitesRequest ofType CheckStudentSatisfiesEnrollmentPrerequisitesRequest
set Query OCL:"enrollForPresentationRequest.studentIdentifier" equalTo Query OCL:"checkEnrollmentPrerequisitesRequest.studentIdentifier"
set Query OCL:"enrollForPresentationRequest.presentationIdentifier" equalTo Query OCL:"checkEnrollmentPrerequisitesRequest.presentationIdentifier"

requestService checkStudentSatisfiesEnrollmentPrerequisites with checkStudentSatisfiesEnrollmentPrerequisitesRequest yielding Variable checkStudentSatisfiesEnrollmentPrerequisitesResult ofType CheckStudentSatisfiesEnrollmentPrerequisitesResult
choice 
{
if Constraint enrollmentMeetsPrerequisitesMet OCL:"checkStudentSatisfiesEnrollmentPrerequisitesResult.enrollmentPrerequisitesMet = true"
doSequential 
{
...
requestService issueInvoice with issueInvoiceRequest yielding Variable issueInvoiceResult ofType IssueInvoiceResult
{
on FinancialPrerequisitesNotSatisfiedException raiseException
FinancialPrerequisitesNotSatisfiedException
}
...
requestService performEnrollment with enrollRequest yielding Variable performEnrollmentResult ofType PerformEnrollmentResult
create Variable enrollForPresentationResult ofType EnrollForPresentationResult
set Query OCL:"issueInvoiceResult.invoice" equalTo Query OCL:"enrollForPresentationResult.invoice"
...
returnResult enrollForPresentationResult
} else raiseException EnrollmentPrerequisitesNotSatisfiedException
}
4 Discussion of the URDAD DSL

In this section we analyse our meta model’s consistency, assess the sufficiency of the URDAD DSL for traceability, code and test generation, and compare the complexities of URDAD DSL and URDAD UML based approaches.

4.1 Consistency

The URDAD meta model describes and relates the concepts of the URDAD domain of discourse. Similarly, Description Logics (DL) are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a well-structured and formally sound manner. In DL, the general terminology of a domain is contained in the TBox. The contingent knowledge about particular individuals is contained in the ABox. In DL, the fundamental inference relation on concept expressions is the subsumption relation. A special case of subsumption is satisfiability which poses the problem of checking whether a concept description does not necessarily denote the empty concept. For empty concepts the set of individuals of this concept is always empty. We have transformed our URDAD meta model $M_1$ into an ALC(D) ontology using the TwoUse [15] Eclipse plugin. The collection of modelling concepts defined by the URDAD meta model was successfully validated to be satisfiable √.
URDAD $\mathcal{M}_0$ instance models were transformed into ontological instance knowledge based on the $\mathcal{ALC(D)}$ representation of the URDAD meta model. The $\mathcal{M}_0$ assertional knowledge and the terminological axioms representing the $\mathcal{M}_1$ model were automatically checked by a Description Logics reasoner to be non-contradictory √. In particular, qualified cardinality restrictions were used to check minimum-cardinality constraints defined for URDAD modelling constructs like that a requirement is required by one or more stakeholders (the unique name assumption is enforced for ontological individuals to prevent the reasoner from inferring their identity).

4.2 Sufficiency for MDE

Requirements traceability is important for design validation and estimation. Validation includes assessing sufficiency and necessity, i.e., assessing whether all requirements are met and whether all model elements are required. This has to be done across levels of granularity [4], whereby four types of traceability should be taken into account [16]: satisfaction links, evolutional links, rationale links, and dependency.

In our meta model, satisfaction links are represented by $\text{usedToAddress}$ links between services and functional requirements. They can be used to trace that all functional requirements are addressed and that one does not use services which do not address functional requirements. Evolutionary links are not addressed within the URDAD meta model as they are provided by the version control environment. Rationale links are also not represented in URDAD as URDAD does not currently include the concept of a goal. The meta model does, however, include $\text{requiredBy}$ links between requirements and their stakeholders, i.e. between a requirement and its source. Dependencies links between model elements are explicitly represented.

The meta model was found to be sufficient to generate or specify a concrete textual grammar capturing the concepts required by the URDAD methodology. The generated editor and parser as well as the standard model validators provide basic validation against the meta model, including compliance to the meta model constraints. We populated an example model and encoded it in both, the URDAD DSL using our textual syntax, and in UML.

The example model has been analyzed by developers and was found to be sufficient for implementation mapping (code generation) and test generation. In addition, a requirements model specified in the URDAD DSL can be mapped onto a process specified in URDAD UML. It does, however, require the mapping of functional requirements onto functional test processes. Further URDAD DSL tool support needs to be developed to facilitate industrial adoption of the DSL. The map-ability from our DSL into UML can facilitate further tool-support.

The concept of an ‘event’ is not reflected in URDAD’s meta model. However, events do not fit naturally into a service-oriented approach where services are regarded as stateless. Even though events can be ‘simulated’ by mapping them onto either the receipt of a service request or a response, URDAD and its meta model are not particularly suited to modelling event-centric systems.
4.3 Complexity

The term ‘complexity’ is here understood intuitively as ‘difficulty of application’ from a practical perspective. When looking at a modelling task, we are confronted with the complexity of the model under construction, the complexity of the language used to this end, and the complexity of the workflow through which the task can be accomplished.

The conceptual model complexity is the same, irrespective of whether the models are denoted in the UML or the URDAD DSL. This is so because we are specifying in both cases the same information about a system under construction.

The language complexity can be assessed by assessing the complexity of its meta model [13]. The language complexity affects its learnability and the complexity of the tools developed around the language including model editors, transformation components and validation tools. Even though a more complex language generally entails a steeper learning curve, it need not result in more complex model encodings. Often the converse is true. A more complex language may have more expressive power and can thus yield smaller model sizes.

The aim of a domain specific language, on the other hand, is to provide a simpler language which introduces only the concepts required for a particular domain, whilst providing the expressive power to effectively make the statements required for that domain, i.e. to be both, a simpler language and to result in more simple models. In the case of our meta model, its UML representation contains about 16 times as many classes and 7 times more relationships than its equivalent in URDAD DSL.

Comparing the model complexity of the URDAD UML model for our example to an equivalent URDAD DSL model we find that the URDAD DSL model has notably lower complexity. Moreover, the URDAD DSL can elegantly describe additional model constructs, which cannot elegantly be represented in URDAD-UML. This is due to the URDAD DSL directly supporting the required semantics for the URDAD methodology, whilst in UML some of the concepts are not directly represented and need to be assembled from more basic model constructs.

The usage complexity will depend largely on the diagrammatic syntax tool support. However, it is expected the lower language complexity, the direct representation of core concepts like services contracts and the much lower level of inconsistency risks will contribute to a lower usage complexity.

5 Related work

The URDAD methodology provides a services-oriented methodology for generating a semi-formal analysis and design model representing MDA’s PIM and supporting test and implementation generation. [11] discuss an alternative approach. Business Rules are specified using OMG’s Semantics for Business Vocabulary and Rules (SBVR). Using MDA tools, these are mapped onto service specifications and orchestration and BPEL process specifications. Processes are
thus assembled from services which are related to business rules. This is similar to our satisfiability links specifying the services used to realize the different functional requirements.

The URDAD DSL allows for the specification of textual and graphical grammars through which the URDAD model is populated. An alternative approach is to define a separate metamodel for the use case narrative and to transform the narrative use case model onto a UML based requirements model [10, 14]. This approach introduces the complexities of having two languages with potential inconsistency risks and an additional model transformation. It thus requires extensive consistency checks between the narrative and the UML models.

[1] stress the need of modelling in the problem domain as well the benefits of accumulating requirements within a single model. Services are grouped into feature sets which are related to responsibility domains. Functional requirements are decomposed across levels of granularity and higher level processes are orchestrated across lower level services. They define the notion of functionals with cause and effect which can be related to the concept of a services contract. In addition they provide a topological functional model (TFM) for mapping technology neutral service requirements onto the available concrete services pool. The TFM and in particular the mapping of service requirements onto concrete services is independent of the modelling technique and can be applied to an URDAD model.

In the Requirements Driven Design Automation methodology (RDDA) [3] one encodes requirements specifications in SYSML diagrams. The SYSML model is enriched with semantic descriptions after which the model is transformed to the One Pass to Production (OPP) design language, the ODL. ODL is an OWL based ontology from which the requirements are validated for consistency and completeness. The approach is, however, structure focused with little emphasis on services contracts and recursive orchestration of higher level services from lower level services.

6 Conclusions and Future Work

We presented in this paper a DSL supporting the model constructs required by the URDAD methodology. The URDAD DSL is substantially simpler than UML and provides better support for service contract specification. Its metamodel has been shown to be consistent. URDAD DSL models support traceability and have notably reduced design complexity. We were able to specify a textual grammar and generate useful language-aware editors and parsers for the practitioner.

The URDAD methodology and DSL represent a services-oriented approach that is not very suitable for event-centric systems. Moreover, the current textual syntax is still too technical for many industrial practitioners. A critical aspect for the success of the URDAD DSL is the definition of a usable graphical syntax and the development of corresponding modelling tools which makes the language more accessible for industrial practitioners. Such tool-support is currently in development. Once this has been done, we plan to conduct an empirical usability
and productivity assessment of the URDAD DSL (compared to an URDAD UML) approach. The derivation of test cases from URDAD DSL specifications is also planned.

Acknowledgments. We thank the national research foundation (NRF) of South Africa for financial support.

References


