A Metamodel Supporting URDAD

Fritz Solms Craig Edwards Stefan Gruner
Alexander Paar
Department of Computer Science, University of Pretoria, South Africa

February 13, 2011

Abstract

URDAD is a methodology which is used by domain experts (e.g. business analysts) to develop requirements models. Historically these requirements models were encoded in UML. However, in order to encode an URDAD model in UML, one has to enforce a strong discipline on the usage of UML. Using UML requires that business analysts go through the mundane tasks of adding a range of semantic relationships in order to fully capture the URDAD semantics. The resultant process is inefficient and error prone. To enforce model quality for a UML encoding of an URDAD model one needs to specify an URDAD UML profile introducing certain URDAD-specific semantics and extensive OCL constraints which constrain the UML model to a valid, consistent and ultimately complete URDAD model. This paper presents the alternative approach of introducing a domain-specific language supporting the URDAD methodology. We introduce and assess an URDAD metamodel which supports URDAD model encodings with significantly lower language and model complexity. Model validation for completeness and consistency as well as URDAD tool development are simplified. We also present a simple concrete text syntax for URDAD which captures the URDAD semantics in a normalized way.

1 Introduction

Requirements quality is still the leading cause of system errors [7]. Besides intrinsic uncertainty and misunderstandings of the requirements by domain experts, some of the core factors which contribute to defects in the captures requirements specification include using specification technologies with informal or vaguely defined requirements semantics [6], specifying requirements only at a single or insufficient level of granularity with a lot of the lower level detailed business requirements being left to the technical team, the inability to validate requirements for

consistency and completeness [**?**], and the inability to test whether a solution fulfils the requirements (non-testable requirements) [3].

Model-Driven Engineering (MDE) aims to address some of these issues by using modeling languages with defined semantics that enforce a level of consistency across model elements, by requiring that the primary model should be a domain model and not a technical model, and by providing a set of tools which support MDE processes. The technology neutral domain model is to be developed by domain experts and not by technical experts [2]. In the case of an enterprise system development domain experts are represented by business analysts.

The *Object Management Group* (OMG) is an industry standards body which has, in the context of its *Model Driven Architecture* (MDA) [8], introduced and standardized a number of technologies, which aim to practically facilitate *Model-Driven Development* (MDD). These include the *Meta-Object Facility* (MOF) for specifying modeling languages, the *Unified Modeling Language* (UML) as a candidate for a generic modeling language, the *Object Constraint Language* (OCL) used to specify constraints at the metamodel or model levels, *Query View Transformation* (QVT) for model-to-model transformations and *MOF Model to Text Transformation Language* (MOFM2T) for generation of textual artefacts such as source code or documentation.

URDAD, the *Use-Case, Responsibility-Driven Analysis and Design* [9] methodology is used by business analysts to develop and validate a technology neutral requirements model and to generate text-based documentation from the domain model. The requirements model can be viewed as the *Platform Independent Model* (PIM) of the *Model Driven Architecture* (MDA) [10].

URDAD is fundamentally a semi-formal services-oriented methodology, which recursively decomposes requirements for a service into a tree of lower level service requirements. Thus, instead of having service requirements at a single level of granularity with complex pre- and post-conditions, the requirements are made more manageable through recursive decomposition into lower level services, which have their own pre-and post conditions and the specification of how processes are orchestrated across the lower level services. Note that, even though it is compelling to mirror the service requirements with corresponding service implementation during the implementation mapping phase, it is important to realise that the recursive decomposition of service requirements is purely a way to manage requirements complexity, make requirements more manageable and facilitate the reuse of requirements components. One could, in fact, roll up the pre-and post conditions across the lower level of granularity services and the corresponding process requirements into a single, flat level of granularity resulting in more complex pre-and post-conditions at a single level of granularity.

The URDAD methodology is independent of the technology used

to encode the semantics of an URDAD model. Working within MDA technologies, one has intrinsically three options for encoding an URDAD model. Traditionally domain experts encoded URDAD requirements models using a standard UML tool enforcing model structure and qualities through an URDAD profile. UML is a natural choice due to its wide spread adoption in industry and strong tool support. However, this approach puts a lot of burden on domain experts to work within the subset of UML required by URDAD and to feed in the appropriate semantics into the model. It also requires an URDAD profile with extensive OCL constraints which constrain the resultant domain model to a valid URDAD model, though the bulk and complexity of these constraints posed significant challenges. The modeling complexity and the incomplete constraints set resulted in high costs around developing and validating domain models and high model defect densities. At times, even the aim of developing an URDAD model with model integrity was abandoned.

The second approach is to develop a front end which only allows domain experts to feed in model elements which comply to an URDAD model. The generated and persisted domain model would still be encoded through a UML tool. We have not pursued this approach.

A third approach is to use MOF to define a *Domain Specific Language* (DSL) for URDAD. The domain is essentially that of a technology neutral services oriented requirements model. There are a number of compelling advantages to this approach. Firstly, the resultant language is a fraction of the size of UML and introduces precisely the semantics required by URDAD. This results in significantly lower model complexity and, with the appropriate tools, simplifies the process through which domain experts specify requirements models. Since only the required semantics are introduced, one also needs to specify a much smaller set of meta-model constraints, which can be used to validate the resultant instance models, both for conformance to an URDAD model, as well as for completeness and consistency.

Furthermore, introducing a DSL requires that one specifies concrete text and/or diagrammatic syntaxes for the DSL. In addition one needs to either develop or generate editing/diagramming tools, which capture an URDAD model through a concrete text or diagrammatic syntax. Using a URDAD-specific DSL also requires that domain experts learn both, the new concrete syntax as well as the tool through which the concrete syntax is captured. Finally, it requires code, documentation and test generation tools to be developed for the URDAD DSL. Fortunately the MDA tool suite and a range of support tools built around this tool suite (particularly in the context of the *Eclipse Modeling Project*) make it relatively easy to specify a concrete text and/or diagrammatic syntax for a DSL based on a MOF metamodel, to define model-to-model and model-to-text transformations and to

generate editors for concrete DSL syntaxes.

This paper is structured as follows. In section 2 we provide a compact overview of the URDAD methodology. Section 3 introduces the URDAD metamodel, its core concepts and the rationale behind it. In section 5 we discuss the verification of the internal integrity of the metamodel, the ability to validate URDAD models for completeness and consistency and assess certain metamodel qualities like complexity, traceability, validatability. Section 6 discusses URDAD tools and introduces and example text syntax for encoding an URDAD model. In section 7 we discuss related work whilst section 8 draws some conclusions and discusses the outlook for future work.

2 Overview of the URDAD methodology

URDAD as a methodology supporting MDA

URDAD is an algorithmic, semi-formal methodology for eliciting and capturing service/use case requirements and technology neutral business process designs[10]. The methodology recursively decomposes cohesive units of functional requirements into lower level functional requirements until all requirements are decomposed into functional components which can be sourced from the environment. The core aim is to provide a repeatable engineering process for eliciting and capturing requirements.

The process is an iterative process, iterating across levels of granularity. At each level of granularity the required functionality is decomposed into cohesive lower level functional units called services. The required logic for assembling the higher level functionality/service from lower level services is specified in a process.

The steps at any level of granularity include

1. lower level functional requirements elicitation

   Decompose the higher level functional requirement (service) into lower level functional requirements required by different stake holders.

2. Specify the services contract for the higher level service including the pre- and post-conditions, data structures for the result and the quality requirements

3.

Discuss recursive nature of URDAD, i.e. how URDAD can be used to design itself but refer to quality paper - feed that into quality paper.

3 The URDAD metamodel

The URDAD metamodel provides the semantics required by the URDAD methodology to store the requirements for a service/use case. The semantics can be encoded in a variety of technologies including ontology based technologies like OWL and object-oriented technologies like MOF. Standard mappings between these two technologies exist [] and we have performed a mapping onto OWL to be able to use standard reasoning services in order to proof the logical consistency of the metamodel.

3.1 Semantics required by URDAD

aim of metamodel -¿ introduce required semantics, minimal, enforce compliance through enforced structure, absorb model constraints as constraints on metamodel

semantics required by URDAD

service = cohesive unit of required functionality

Metamodel = services requirements recursively decomposed into lower level services requirements with pre- and post-conditions as well as quality requirements for each service and the data structure specifications for the inputs and outputs and the process specification on how a service is orchestrated across lower level services.

post-conditions may change environment -¿ inverse services

Fixing of levels of granularity

responsibility domains as roles and packages

functional and quality requirements can be reused across services.

data types and data structures

expressions including constraint expressions

3.2 The metamodel

Proposed structure Ideas for issues to be covered. Feel free to shuffle, remove, adjust, reword, mock, critique, hide, destroy etc References/related work still needs to be added Statements will need to be sanity checked

Fritz: Need to check overlap between this section and introduction

- URDAD META MODEL JUSTIFICATION
  - In an effort to formalise URDAD, there exists a need for the formal specification of URDAD's semantics.
  - The concepts and rules associated with the use of the URDAD methodology must be clearly defined.
  - The creation of an URDAD metamodel will assist with the formal definition of URDAD's semantics.

- Define Metamodeling
- Models versus metamodels. Matter of perception. Discuss levels of abstraction. (One particular metamodel may be considered to be an instance model of another more abstract metamodel.)
- A metamodel represents a level of abstraction.
- Metamodels are closely related to ontologies. (Concept definition, depiction, including relationships between concepts) DISCUSS
- While the concepts associated with URDAD are independent of any particular technology or organisation, URDAD is currently used to produce Platform Independent Models (PIMs) as envisaged in Model Driven Architecture (MDA), the Object Management Group's (OMG) approach to Model Driven Engineering.
- PIMs are independent representations of processes, which may or may not be implemented as a software system. They are used to depict the functional requirements of a system. Once all non-functional requirements have been taken into consideration and an appropriate implementation environment has been selected, PIMs can be used to create Platform Specific Models (PSM) through a process of model transformation and refinement. Unlike PIMs, PSMs are inseparable from the actual technology platforms that will be used to implement the system.
- Historically UML has been used to encode URDAD models.
- UML was a natural selection considering that it represents a standards-based modelling language that is managed by the OMG.
- Unfortunately encoding the URDAD model in UML is not a trivial exercise.
- It is difficult to ensure that the model is encoded in a consistent manner and it requires a strong discipline in the usage of UML.
- For example, actors are used to represent the stakeholders in UML use case diagrams. URDAD prescribes the use of interfaces to depict the roles fulfilled by stakeholders.
- For each service URDAD requires that there is a clear linkage between the service's stakeholders and their functional and quality requirements. (REF URDAD paper)
- An URDAD UML profile was created in order to help ensure that URDAD UML models are consistently encoded.

  - For example, the URDAD profile introduced a ¡¡requires¿¿ dependency which facilitated the linkage between a service's stakeholders and their requirements.
  - However, while this profile has helped improve the quality and consistency of URDAD models encoded in UML, the process is still error prone and unintuitive.
  - Domain experts tasked with creating URDAD UML models are forced to add a range of mundane relationships and stereotypes in order to accurately capture URDAD's semantics.
  - It is also not trivial to capture certain aspects of URDAD's semantics in UML, even with the assistance for a UML profile. (Examples?)
  - The creation of an URDAD metamodel attempts to address these issues.
  - Depending on the manner in which the metamodel is encoded, URDAD models that conform to the metamodel will require substantially less manual validation. These models will also hopefully be less complex, especially if tools are developed to support the creation of these models.
- CONCEPTS THAT DEFINE URDAD'S METAMODEL
  - The URDAD metamodel and the semantics it seeks to define, are independent of its physical encoding.
  - URDAD's metamodel can be encoded in more than one format.
  - When selecting a mechanism to encode the metamodel, it is critical that URDAD semantics are represented in their entirety. If this is not the case one could argue that the encoded metamodel does not truly represent URDAD. *Fritz: What is meant by this?*
  - The fundamental concepts that define the URDAD metamodel are as follows:
    * THE MODEL
      · Represents the requirements model instance and it constituents
      · A model represents the formal definition of requirements for a problem domain, modeled as services within responsibility domains, where each service is constrained by the functional and quality requirements of its stakeholders. The stakeholders are themselves represented as responsibility domains.
    * RESPONSIBILITY DOMAINS

- Represents a logical grouping of elements within a model, where each element belongs to the responsibility domain.
- Similar to the concept of packaging, but conceptually more consistent with the responsibility driven nature of URDAD.
- Responsibility domains also provide a consistent mechanism of depicting the roles/stakeholders within the model.
- Responsibilities domains may only be composed of data structures, services, requirements and other responsibility domains.
- Each role is associated with a cohesive list of responsibilities.
- The introduction of the concept of a responsibility domain eliminates the need for the separate definition of a services contract, which is traditionally used to group logical related services. Services are now grouped by the responsibility domain within which they reside.
- In the tradition of namespaces each element within a responsibility domain is uniquely identified by its name appended to the fully qualified name of the domain itself.

* SERVICES
* Each service exists to realise a use case *Fritz: at some level of granularity the user might be a higher level service, maybe we want to avoid the use case vs service mess and focus on using services* and fulfil the requirements of its stakeholders.
* Users of a service are also represented as stakeholders.
* Services represent a level of granularity. Each level of granularity can be regarded as a level of abstraction.
* Requirements exist at a particular level of granularity, and are themselves decomposed further across subsequent lower levels of granularity.
* There are two forms of requirements that a service seeks to address namely functional and quality requirements.
* It is important to note that both functional and quality requirements are not restricted to a single service. There exists the reality that more than one service may have requirements in common.

* URDAD does not address any non-functional requirements other than the non-functional requirements of the requirements themselves. For example, requirements should be easy to maintain. Requirements should exhibit principles of good design, such as decomposition across layers of granularity, single responsibility, loose coupling and high cohesiveness.
* Each service is represented as a formalised contract, with explicit pre and post conditions that seek to address the functional requirements of the service.
* Pre-conditions represent the conditions under which the service may legally be refused. A pre-condition's existence must be justified by a tangible functional requirement.
* A pre-condition is contractually obligated to raise an exception if it is not fulfilled. This exception must be clearly specified on the service contract.
* Post-conditions represent the conditions that must hold true if all the pre-conditions have be fulfilled and the service is complete.
* Unless the service has been sourced from the environment, it must be possible to verify whether each post-condition has been fulfilled. (VERIFY THIS STATEMENT)
* The fulfilment of a post condition may have a lasting impact on the state of the environment. Inverse services may be associated with each post-condition. These services are responsible for reversing the effects the post-condition had on the environment and returning the environment to its original state, in the event of an error occurring during the execution of the service.
* Associated with each service there exists a process definition that is composed of all the activities that are required to provide the service. Each activity must either directly or directly be associated with:
  - The validation of a pre-condition through the execution of a lower level service.
  - The fulfilment of a post-condition through the execution of a lower level service.
  - The simultaneous validation of a pre-condition and fulfilment of a post-condition; through the execution of a common lower level service.
  - The construction of the service result

        * The process defines the logical orchestration of the activities that are required to provide the service.
        * The lower level services utilised by each activity constitute the next level of granularity, when considered in the context of the particular service in question.
        * All activities within a process must directly address one or more of the functional requirements associated with the service.
        * No activities should exist that do not address functional requirements.
        * An important non-functional requirement of all activities that constitute a service, is that each activity must be able to be traced back to the fulfilment of a functional requirement. There should be no redundancy.
        * Each service has a consistent service signature. The signature takes the form of an appropriately named service, a single service request object that contains detailed information pertaining to the request and a single result object, which is composed of the information associated with the result of the service execution.
        * Traceability is one of the essential characteristics of the service contract and its process definition. Traceability is realised as follows:
            · Ensuring that activities only exist to address a pre-condition, a post-condition or both a pre-condition and a post-condition simultaneously.
            · Associating each functional requirement with one or more stakeholder's represented as responsibility domains.
            · Ensuring that a service addresses all functional requirements and only the relevant functional requirements.

- METAMODEL ENCODING OPTIONS
    - There are many languages and approaches that can be used to encode the URDAD metamodel.
    - It is important that the advantages and disadvantages of each encoding option are carefully taken into consideration.
    - Regardless of the encoding option that is ultimately selected, URDAD's semantics must be able to be represented in their entirety.
    - Each encoding may differ in the way in which URDAD's semantics are depicted.

- (INSERT REF - Quality in Model-Driven Engineering) argues that model quality is determined by five aspects. The modeling language and tools used to define the model, the modeling process itself, techniques used to assure quality and the relative experience of the individuals tasked with building the model.
- An encoding option should be assessed according to these five aspects.
- Three encoding options have been considered. Encoding options are not mutually exclusive and several may be concurrently utilised if each offers its own unique advantages. For example, one particular encoding option may make it easier to represent semantics and reason about the model, while another encoding option may be aligned with standards and offer extensive tool support for model tool smiths and practitioners.
- Currently three encoding options have been taken into consideration.
- The first option is to attempt to improve the UML encoding of URDAD. While this will always be an option worth considering, it was quickly discarded for the reasons already mentioned in this paper.
- Another encoding option that has been considered is the encoding of the URDAD metamodel in a knowledge representation language such as the Web Ontology Language (OWL) or more specifically OWL-DL one of OWL's sub languages. There are many benefits associated with this option... (ELABORATE)
- The third opinion and the encoding which represents the focus of this paper is to encode the URDAD metamodel using the Eclipse Modeling Framework's (EMF) Ecore metamodel.
- It is possible to capture URDAD's metamodel by extending the Ecore metamodel and introducing URDAD's semantics.
- The Eclipse
  TODO COMPLETE...
  Discuss the tool support Discuss Ecore and in particular the Eclipse Modeling Project's alignment with the OMG standards. Natural selection considering URDAD current alignment with the OMG's MDA Ecore ¡-¿ EMOF OCL QVT M2T Textual and graphical concrete syntaxes (XText and EMF Text) Model Validation IDE environment - services offered (syntax checking etc)
- One may consider the formalisation and encoding of URDAD's metamodel to be an attempt to establish a Domain

Specific Language (DSL) for URDAD by defining its abstract syntax.

- THE ECORE METAMODEL ENCODING
  TODO COMPLETE...
  Discuss how the URDAD metamodel is realised in Ecore
- OUTSTANDING ISSUES AND POSSIBLE IMPROVEMENTS TO THE ECORE METAMODEL
  TODO COMPLETE...

4 Example model

Compare to "handle claim" in [1]

5 Assessing the URDAD metamodel

5.1 Core metamodel integrity assessment

- Check that no classes unsatisfiable
- Check consistency of OCL constraints
- Redundency checks (redundent classes, redundent attributes and properties) redundend constraints

5.2 Assessing model qualities

- Comparing complexity of URDAD-DSL and UML encodings of URDAD model
  - UML model requires much more model elements to encode URDAD semantics.
- Assessing traceability ([5] - see notes on Zotero entry)
- Completeness checks (What does completeness mean - That only technical information needs to be provided and that the full requirements from a business perspective are specified across levels of granularity.)
  - all OCL constraints for decision conditions specified
  - all OCL constraints for pre and post-conditions specified
  - All required request and result fields all specified either via OCL constraints or by default values.
- Check that process addresses all functional requirements and nothing but the functional requirements.

5.3 Usability assessment (potentially an empirical study)

- Have 2 groups doing URDAD modeling, one using UML encoding and UML tools and one using URDAD encoding.
- Assess relative productivity, error densities and completeness of the two resultant models

6 Tools

7 Related work

- [1] discuss the mapping of business rules specified using OMG's *Semantics for Business Vocabulary and Rules* (SBVR) to service specification and orchestration, mapping onto BPEL process specifications via MDA tools.
- [2] stress the need for performing the modeling in the problem domain as well as the need to accumulate the requirements within a single model. They effectively also group services into responsibility domains represented by their notion of feature sets, decompose functional requirements across levels of granularity, orchestrate higher level processes across lower level services and define the notion of functional with cause and effect which can be viewed as a way of specifying a services contract. In addition they provide a *topological functional model* (TFM) for mapping technology neutral service requirements onto available concrete services pool. The TFM is independent of the modeling technique and can be applied to an URDAD model
- [4] discuss the *Requirements Driven Design Automation* methodology (RDDA) for requirements specification which are encoded SYSML diagrams to which semantic descriptions are added. The model is transformed to the *One Pass to Production* (OPP) design language, the ODL, which is an OWL based ontology from which the requirements are validated for consistency and completeness. Their approach is structure focused, putting little emphasis on services contracts and recursive orchestration of higher level services from lower level services.
- [3] stress need for testable domain models. Instead of defining services contracts explicitly via pre- and post-conditions, they generate the effective services contracts from process specifications and subsequently extract tests from the generated contracts.

8 Conclusions and outlook

References

[1] A Model-Driven perspective on the Rule-Based specification of services. Munich, Germany.

[2] Erika Asnina and Janis Osis. Computation independent models: Bridging problem and solution domains. Athens, Greece, 2010.

[3] Soheila Bashardoust-Tajali and Jean-Pierre Corriveau. On extracting tests from a testable model in the context of domain engineering. In *13th IEEE International Conference on Engineering of Complex Computer Systems (iceccs 2008)*, pages 98–107, Belfast, Northern Ireland, 2008.

[4] I. Cardei, M. Fonoage, and R. Shankar. Model based requirements specification and validation for component architectures. In *Systems Conference, 2008 2nd Annual IEEE*, page 18, April 2008.

[5] Jeremy Dick. Design traceability. *Software, IEEE*, 22(6):1416, November 2005.

[6] Robert Ferguson and Giuseppe Lami. An empirical study on the relationship between defective requirements and test failures. In *2006 30th Annual IEEE/NASA Software Engineering Workshop*, pages 7–10, Columbia, MD, USA, April 2006.

[7] Petra Heck and Pivi Parviainen. Experiences on analysis of requirements quality. In *2008 The Third International Conference on Software Engineering Advances*, pages 367–372, Sliema, Malta, 2008.

[8] Jon Siegel. Developing in OMG's Model-Driven architecture. White paper, Object Management Group, November 2001.

[9] Fritz Solms. Technology neutral business process design using URDAD. In *Proceeding of the 2007 conference on New Trends in Software Methodologies, Tools and Techniques*, page 5270, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[10] Fritz Solms and Dawid Loubser. URDAD as a semi-formal approach to analysis and design. *Innovations in Systems and Software Engineering*, 6:155–162, 2010. 10.1007/s11334-009-0113-4.