URDAD as a Semi-Formal Approach to Analysis and Design

Fritz Solms and Dawid Loubser
Solms Training, Consulting and Development
113 Barry Hertzog Ave, Emmarentia, Johannesburg, 2915, South Africa
www.solms.co.za, fritz@solms.co.za, dawidl@solms.co.za

***Abstract*—The Use Case, Responsibility Driven Analysis and Design (URDAD) methodology, is a methodology for technology neutral design generating the Platform Independent Model (PIM) of the Object Management Group's (OMG's) Model Driven Architecture (MDA). It requires the core modeling to be done in the problem space by domain specialists and not in the solution space by technology specialists. URDAD allows for formal elements to be added by different role players at different stages of the model refinement, whilst aiming to preserve agility of the outputs and low cost of the process generating the outputs. This paper discusses the semi-formal aspects of URDAD which facilitate modeling validation and testing, documentation generation and automated implementation mapping as well as aspects which promote agility and low cost.**

***Index Terms*—RDAD, formal methods, agile processes RDAD, formal methods, agile processes U**

I. Introduction

Formal methods aim to provide non-ambiguous requirements and a design which can be proven to be correct and reliable [11]. These methods, however, are typically perceived as difficult and expensive: their use has historically largely been confined to a relatively small class of problems where safety and reliability are paramount (such as the aviation [4] and defense industries, where testing and debugging costs can contribute more than 50% of the development costs [13].)

Agile methods [2], [9], on the other hand, accept that the initial requirements are potentially incomplete, partially incorrect and volatile. They aim to provide a process which can operate successfully within such an environment. Modeling is seen largely as a tool used to facilitate domain exploration and simplification of the solution whilst the primary output of the process is working code.

Model-driven development (MDD) [18], [14] has been influenced by both, formal [12] and agile [8], [17] methods. The formal approach is required for model transformation and code generation. The agile aspect is important for business and system agility, i.e. to cost effectively address requirements evolution.

The OMG's vision of MDD is MDA [15]. It is supported by an elaborate technology suite including support for modeling via the Unified Modeling Language (UML), meta language specification via the Meta-Object Facility (MOF), object constraint language (OCL) and the Query-View Transformations (QVT) specification for querying and transforming models.

The value obtained from MDA based projects has been limited by [17]

- **the lack of standards for specifying the implementation architecture and technologies,**
- **the lack of an well-defined analysis and design methodology used to generate the platform independent model (PIM), and thirdly by**
- **the wide variation in model structure and content permitted by the UML, increasing the complexity of the implementation mapping.**

URDAD aims to address the latter two by providing

- **a step-for-step algorithm for performing technology-neutral analysis and design, with defined inputs and outputs for each step, and**
- **restricting the use of UML constructs to a small but sufficient subset of UML, and enforcing a specific model structure with specified model elements (via a set of OCL-based model validation suites.)**

In this paper we point out the semi-formal aspects of URDAD facilitating model validation, automated generation of functional testing, documentation generation and simplified implementation mapping as well as aspects of an URDAD model and approach which improve agility.

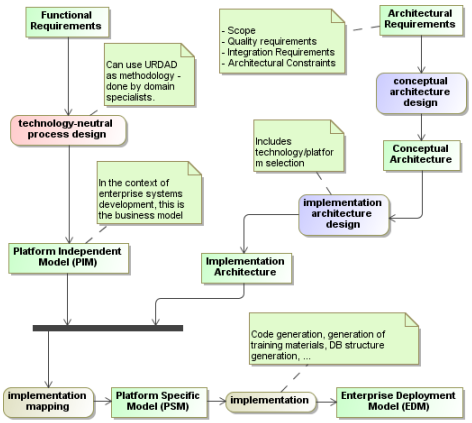

Fig. 1. Separation of functionality and architecture/technologies as envisaged by the MDA.

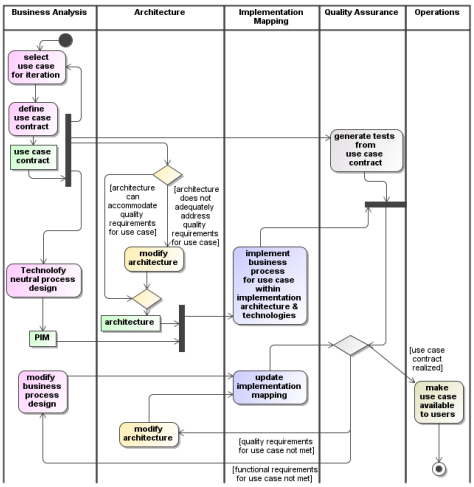

Fig. 2. Separation of concerns in a typical model-driven developmen process.

II. URDAD in the context of Model-Driven Development

In model-driven development (MDD) [18], [14], and the Object Management Group's (OMG) Model Driven Architecture (MDA) [18], [14] in particular, the core artifact is the model which specified in terms of concepts from the problem domain instead of the solution domain. The model thus needs to be generated and maintained by domain experts (e.g. business analysts in the context of enterprise systems development) and not by technical specialists.

The model contains the functional requirements, together with the processes and other design elements (such as the data structures required by the processes) through which the functional requirements are realized.

The information about the technical infrastructure within which the model needs to be implemented is provided by a separate architecture/technologies specification, which can vary independently from the functional model. The architecture needs to be designed such that the infrastructure is able to address the non-functional requirements of the services (e.g. performance, scalability, reliability, security, auditability, accessibility, usability, ...). In the context of enterprise architecture, the architecture spans across organizational and systems architecture, providing the organization with a suitable infrastructure within which it can realize its vision and mission, and within which business processes are deployed and executed. It seamlessly encompasses software, hardware and human components.

Figure 1 shows MDA's envisaged separation of architecture/technology from process design. Once the process addressing the functional requirements has been designed, the implementation mapping of the resultant platform independent model (PIM) into the architecture designed to address the quality requirements can be automated using MDA tools. The tools can also potentially be used to perform other transformations like that of transforming the URDAD model into a formal methods-based representation, like a Petri Net or documentation generation.

One of the aims of model-driven development is the clean separation of responsibilities across architecture, (business) analysis, implementation mapping, quality assurance and operations. A typical model driven development process is shown in figure 2.

The domain/business analysts perform the stake holder requirements elicitation, validation and documentation resulting in a use case or services contract. This contract is used by quality assurance (QA) to (auto)-generate the functional and non-functional tests. Architecture assesses whether the current infrastructure can host the service with its particular quality requirements and, if required, makes the necessary architectural modifications. Domain analysts then perform the technology neutral process design. It is for this step that URDAD is used. Once the technology neutral design is complete and once architecture has signed off the architecture specification, the implementation mapping is done after which QA will have to assess whether both, the functional and non-functional requirements are fulfilled. In the case where there are issues with the functional requirements (i.e. with what has or has not been done), the error reports are provided to domain analysis which then either makes the required modifications to the technology neutral design or notifies implementation mapping that certain mappings have been implemented incorrectly. In either case the implementation mapping is updated after which it goes back to QA.

On the other hand, should QA finds quality of service issues (e.g. performance, security, reliability, scalability, ..., issues), the problem is reported to architecture which either makes appropriate architectural modifications or notifies implementation mapping of concerns with the implementation. Once QA has verified the realization of the use case contract, the service is handed over to operations to be made available to the users.

III. The URDAD methodology

URDAD [17], [16] is an analysis and design methodology which has been adopted by a number of organizations, particularly in the financial and insurance sectors, for their technology neutral business process design [7]. It aims to provide a simple repeatable engineering process for generating generates MDA's PIM, in the sense that the outputs from different domain experts with similar domain knowledge would be very similar. To achieve this, URDAD provides an algorithmic process for analysing and designing the technology neutral solution.

The methodology specifies a mechanism for fixing levels of granularity, and has specified steps (modeling activities) for each level of granularity. The type and structure of the inputs and outputs of each step are precisely specified.

One of the core aspects of URDAD is that it requires analysis and design to be done across levels of granularity, with both of these done by domain experts from different specialization areas. Thus instead of requiring a (business) analyst to specify the entire requirements for a new service, this responsibility is distributed across the domain experts touched by the service requirements.

For example, the business process for processing an insurance claim may require services for assessing the claim coverage and value, as well as for settling the claim and recuperating any losses. Those are very different responsibility domains and it is unreasonable to expect a business analyst to be able to provide the detailed requirements for a business process across levels of granularity. Instead a business analyst would specify the high-level requirements (such as that the losses need to be recuperated) without having to specify the detailed requirements and process for those lower level services. He is, however, assembling the higher level business process across those lower level services.

In particular the methodology requires that the URDAD-compliant UML model is populated through the following views (UML diagrams):

- **A *Services contract view* as a UML class diagram containing the UML interface with the service signature, class diagrams specifying the data structures of the request and response objects, and the pre- and post-conditions and quality requiremens for the service.**
- **A *User workflow view* which is typically a sequence diagram showing the interaction (messages exchanged) between the user and the service provider for the use case. Both the user and service provider are represented by UML interfaces for the contracts which are populated with the services required from these role players.**
- **A *Functional requirements view* which shows the lower levels services required to address the pre- and post-conditions.**
- **A *Responsibility allocation view* which specifies how the services are assigned to services contracts via usage dependency which point from the functional requirement/use case to the services contract which hosts the corresponding service.**
- **A *Process view* specifying the (business) process through which the service is to be realized in the form of a UML activity diagram which is assigned to be the behavior of the service. The input and output parameter nodes correspond to the request and response objects as specified in the services contract for the service. It shows the process flow across call operations requesting the lower level services from which the (business) process is assembled.**

Services are recursively constructed from lower level services with the lowest level services being either not domain specific (generic services like numeric addition, or infrastructure services such as persistence) or services which are sourced externally. For these externally-sourced services, the methodology still requires the specification of a full services contract, but are otherwise handled as black boxes.

IV. SEMI-FORMAL ELEMENTS OF URDAD

Formal methods [11], [5] are precise, mathematically rigorous methods which remove ambiguity and result in verifiable implementations. The three core approaches are based on denotational, operational and axiomatic semantics.

In both denotional and operational semantics, one defines the meaning of what the program represents/does. These approaches provide the ability to formally prove an algorithm. For complex problems the approach may, however, be very costly. Furthermore, there could be multiple algorithms which could suitably realize the stakeholder's requirements - yet one will lock into a particular algorithm which can potentially be proven correct, but which may not be the best with respect to certain quality requirements like simplicity, performance, resource-efficiency (e.g. memory efficiency, ...) and so on. Furthermore, replacing one algorithm with another is a complex task which typically requires a complete redefinition of the algorithm and its proofs, and may have a wide ranging impact across the system.

In axiomatic semantics (also known as contract-based approaches) one does not provide the formal specification of a program, but instead the formal specification of the effect the program needs to have on its environment. This can be done in the form of a contracts specified via pre- and post-conditions using constraint languages like OCL [3]. In this approach, algorithms are proven against the contract they realize, either by formally proving that a particular algorithm realizes the contract, or by generating algorithm-independent tests from the contracts [10]. In the former approach this is usually done recursively, by assuming at any particular level of granularity, that the lower level services used within the algorithm do realize their respective services contracts. In the case of generating algorithm-independent tests from the contracts, one also needs to generate a complete set of test data which covers the entire application domain. This may be impractical or even impossible. URDAD can be seen as an analysis and design methodology which enforces a contracts-based approach across levels of granularity, encouraging the formal contract specification in the form of OCL constraints.

An axiomatic/contracts based approach is more natural to model-driven development, since the model should preferably be developed in the problem/requirements domain and not in the technical/solution domain. Contracts are independent of particular service providers, or the algorithms they use.

However, the requirements specification in the form of formal contracts is often non-trivial. A complete contract for a high-level service may be very complex. Domain experts (e.g. business analysts) who develop the model require a framework within which they can incrementally refine the requirements, and the technology neutral solution (business processes). They typically do not have sufficient technical skills to formally define the services contracts in OCL. URDAD encourages them to define the services contracts informally (using natural language to specify constraints, for example), with OCL experts subsequently mapping such statements to OCL.

In addition to the service contracts across levels of granularity, URDAD enforces a number of other elements which formalize the outputs of the methodology including

- **an OCL validation suite which validate that the UML model complies to the constraints of an URDAD meta-model, and**
- **OCL validation suites for model completeness and consistency.**

These validation suites enforce

- **the linking of pre- and post-conditions to functional/service requirements (use cases) through which these pre- and post-condition is addressed,**
- **the linking of functional requirements (use cases) to services in services contracts which formalize the service requirements,**
- **the linking of service process specifications to the service contracts they realize,**
- **the construction of request objects for each service from the information available to the (business) process at the stage of service request,**
- **the construction of result objects from the information available at the end of the process realizing the service,**
- **the specification of contracts for all role players including users and service providers across levels of granularity,**
- **that for each pre-condition a corresponding exception is raised, informing the user that the service requested is not going to be provided. All other scenarios satisfying the pre-conditions for the service lead to the provision of the result object.**

The first three provide bi-directional requirements traceability. Black-box tests can be auto-generated from the services contracts.

Note that if the levels of granularity are not carefully managed, the specification of some of the above constraints can become very complex. The methodology, however, aims to provide a repeatable algorithm through which the levels of granularity are fixed in a way which minimizes complexity at any particular level of granularity, and which maximizes re-use of services.

The validation suites enforces a complete, consistent model which has sufficient information to perform a complete implementation mapping, provided the semantics for the implementation architecture and technologies are provided.

V. What about agility?

Agile approaches [9], [2] like Extreme Programming [1] accept and even welcome continuously changing requirements in the context of continuously changing business opportunities and continuous growth of knowledge on how better to generate stake holder value. They aim to be able to provide better business value by being able to effectively operate in an environment of continuously changing requirements.

In such a high-risk environment it is critical to manage and mitigate risk. This is done through practices like short feedback cycles via short releases, on-site customer, pair-programming, enforced, automated functional (unit) testing across levels of granularity, and continuous integration testing.

Agile methodologies typically handle the removal of any unnecessary complexity arising from an agile approach through peer reviews and continuous refactoring facilitated through non-ownership of all artifacts generated by the process.

A. Applying agile principles and practices in MDD

MDD decouples the requirements and design from the implementation architecture and technologies and increases the level of abstraction of development. However, continuously changing requirements are core to business agility and many of the agile practices have been ported to this higher level of abstraction including on-site customer (with the customer communication simplified as the design is done in the problem/business domain), pair design, enforced and automated functional testing at the design level either via proving the design or executing the design in model execution environments [6], peer reviews and non-ownership of designs.

In addition, MDD, and MDA in particular, aim to improve agility around technology and architecture by decoupling the design from these and automating the implementation mapping onto the target architecture and technologies.

B. How does URDAD aim to achieve agility?

URDAD itself is not a development process but a process for performing the technology neutral analysis and design generating MDA's PIM. It is typically embedded within a model-driven development process within which some of the agile principles and practices can be absorbed.

However, the URDAD methodology aims to increase agility through a number of intrinsic process elements

- **enforced decoupling across all levels of granularity facilitated through enforced binding to services contracts, implied adapters to concrete service providers and localization of process in separate work flow controllers,**
- **explicit search step for service reuse resulting in lower cost and complexity reduction,**
- **automated documentation (including UML to natural language mapping and UML diagram generation) from the URDAD compliant UML model, and**
- **complexity reduction through enforced responsibility localization.**

VI. Conclusions and outlook

Over the years URDAD has strengthened its formal aspects through model validation and increased enforcing of formal OCL based contracts. This simplifies, model testing and well as model transformation tasks like documentation generation and implementation mappings.

However, using standard UML modeling tools results in a lot of unnecessary overheads for modelers who have to construct the appropriate URDAD model structure themselves, obtaining only guidelines from the results of the model validations. The agility of URDAD can be considerably improved by

- **developing a URDAD front-end to UML which enforces the URDAD model structure directly and which guides modelers explicitly through the URDAD process,**
- **extending the range of documentation generation transformations to provide suitable modelviews for different role players, and**
- **developing URDAD specific implementation mapping transformations for widely used implementation technologies in infrastructures.**

VII. Acknowledgments

We would like to thank Dr Stefan Gruner for the many thought provoking discussions we had on the submitted topic.

REFERENCES

[1] Kent Beck and Cynthia Andres. *Extreme Programming Explained*. Addison-Wesley Professional, second edition, 2004.

[2] Kent Beck, Martin Fowler, Robert C. Martin, et al. The agile manifesto. http://agilemanifesto.org/, 2001.

[3] L.C. Briand, Y. Labiche, M. Di Penta, and H. Yan-Bondoc. An experimental investigation of formality in uml-based development. *Software Engineering, IEEE Transactions on*, 31(10):833–849, Oct. 2005.

[4] A. Hall and D. Isaac. Software in air traffic control systems. In *The Future, IEE Colloquium on*, pages 7/1–7/4, Jun 1992.

[5] Mike Hinchey, Michael Jackson, Patrick Cousot, Byron Cook, Jonathan P. Bowen, and Tiziana Margaria. Software engineering and formal methods. *Commun. ACM*, 51(9):54–59, 2008.

[6] Andrei Kirshin, Dolev Dotan, and Alan Hartman. *A UML Simulator Based on a Generic Model Execution Engine*, volume 4364/2007 of *Lecture Notes in Computer Science*, pages 324–6. Springer, Berlin / Heidelberg, 2007.

[7] Riaan Klopper, Stefan Gruner, and Derrick Kourie. Assessment of a framework to compare software development methodologies. In *Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, volume 226 of *ACM International Conference Proceeding Series*, pages 56–65. ACM Press, 2007.

[8] I. Lazar, B. Parv, S. Motogna, I.-G. Czibula, and C.-L. Lazar. An agile mda approach for the development of service-oriented component-based applications. In *CANS '08: Proceedings of the 2008 First International Conference on Complexity and Intelligence of the Artificial and Natural Complex Systems. Medical Applications of the Complex Systems. Biomedical Computing*, pages 38–44, Washington, DC, USA, 2008. IEEE Computer Society.

[9] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice-Hall, 2002.

[10] Bertrand Meyer, Arno Fiva, Ilinca Ciupa, Andreas Leitner, Yi Wei, and Emmanuel Stapf. Programs that test themselves. *Computer*, 42(9):46–55, Sept. 2009.

[11] Jean Fran?ois Monin and Michael Gerard Hinchey. *Understanding formal methods*. Springer, 2003.

[12] Flavio Oquendo. Pi-method: a model-driven formal method for architecture-centric software engineering. *SIGSOFT Softw. Eng. Notes*, 31(3):1–13, 2006.

[13] A. Platzer. Verification of cyberphysical transportation systems. *Intelligent Systems, IEEE*, 24(4):10–13, July-Aug. 2009.

[14] Sudipto Ghosh Trung Dinh-Trong Robert B. France and Arnor Solberg. Model-driven development using UML-2: Promises and pitfalls. *Computer*, 39(2):59–66, February 2006.

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

[16] Fritz Solms. Technology neutral business process design using URDAD. In Hamido Fujita and Domenico Pisanelli, editors, *New Trends in Software Methodologies, Tools and Techniques*, Frontiers in Artificial Intelligence and Applications, pages 52–70. IOS Press, 2007.

[17] Fritz Solms and Dawid Loubser. Generating mda's platform independent model using urdad. *Knowledge-Based Systems*, 22:174–185, 2009.

[18] Thomas Stahl and Markus Voelter. *Model-Driven Software Development*. John Wiley & Sons, 2004.