Innovations Syst Softw Eng (2010) 6:155–162
DOI 10.1007/s11334-009-0113-4

ORIGINAL PAPER

URDAD as a semi-formal approach to analysis and design

Fritz Solms · Dawid Loubser

Received: 30 November 2009 / Accepted: 4 December 2009 / Published online: 1 January 2010

Abstract The Use Case, Responsibility Driven Analysis and Design (URDAD) methodology is a methodology for technology neutral design generating the Platform Independent Model of the Object Management Group's Model Driven Architecture. 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 model validation and testing, documentation generation and automated implementation mapping as well as aspects which promote agility and low cost.

Keywords URDAD method · Formal methods · Agile processes

1 Introduction

Formal methods aim to provide unambiguous requirements and a design which can be proven to be correct and reliable [14]. These methods, however, are typically perceived as difficult and expensive: historically their use has largely been confined to a relatively small class of problems where safety and reliability are paramount (such as the aviation [6] and defense industries, where testing and debugging costs can contribute more than 50% of the development costs [17]).

Agile methods [2,11], 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) [5,21] has been influenced by both, formal [16] and agile [9,20] 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 [18]. 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), constraint specification, model querying and transformation via the Object Constraint Language (OCL) and the Query-View Transformations (QVT) specification. MDA aims to provide a level of agility whilst being sufficiently formal to be able to automate large parts of the implementation mapping by enabling domain experts to perform a semi-formal analysis and design in UML which can then be incrementally formalized through model refinement and the use of OCL.

The value obtained from MDA based projects has been limited by [20]

– the lack of standards for specifying the implementation architecture and technologies,

This paper is part of an ongoing PhD project by F. Solms under the supervision of Stefan Gruner at the Department of Computer Science of the University of Pretoria. We would like to thank Stefan Gruner for the fruitful discussions we had on this topic.

F. Solms (✉)
Department Computer Science, University of Pretoria,
Pretoria, Republic of South Africa
e-mail: fritz@solms.co.za

D. Loubser
Solms Training and Consulting, 113 Barry Hertzog Ave,
Emmarentia, Johannesburg, South Africa

– the lack of a well defined analysis and design methodology used to generate the platform independent model (PIM), and
– the wide variation in model structure and content permitted by the UML, increasing the complexity of the implementation mapping.

Various attempts have been made to both simplify and increase the formality of UML models by restricting the number of diagrams and UML elements used and adding formal specification techniques to the UML model [4,12]. URDAD similarly restricts the use of UML constructs to a small but sufficient subset of UML, and enforcing a specific model structure with specified model elements. However, with the availability and growing support for the OCL, a lot of formalization is done within UML and OCL.

A number of service oriented design methodologies have been proposed [10,15]. Ma et al. [10], like URDAD, integrates the services oriented approach within a model-driven approach. Many of these services oriented design methodologies use BPMN and often these approaches are technology specific. URDAD aims to provide a more formal, technology neutral approach based on the use of UML and OCL. OCL is used to formalize both the contract and the process specifications. This semi-formal approach facilitates model validation and transformation including documentation and code generation.

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.

2 URDAD

URDAD [19,20] is a service-oriented 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 [8]. It aims to provide a simple repeatable engineering process for generating 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 a step-by-step process for analysing and designing the technology neutral solution with well-defined inputs and outputs for each step.

2.1 The methodology

Instead of following a waterfall approach within a particular service/use case (i.e. performing analysis, design, implementation, testing and deployment for each use case), URDAD requires the one which performs analysis and design across levels of granularity. This is illustrated in Fig. 1 which shows that for any use case, the high-level requirements analysis and formalization within a services contract is followed by a design which assembles processes realizing the higher level service from lower level services for which the lower level analysis and design is done as one traverses to the next lower level of granularity.

Services are recursively constructed from lower level services with the lowest level services being neither domain specific (generic services like numeric addition, or infrastructure services such as persistence) nor 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. Figure 1 also points out that one terminates the analysis and design as soon as all required services are available.

One of the core aspects of URDAD is that it requires analysis and design to be done across levels of granularity, with both 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 from claims could 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/she is assembling the higher level business process across level services sourced from other domains of responsibility whose detailed requirements and designs are done by domain experts from those domains of responsibility.

2.2 The analysis phase

Figure 2 shows the steps for the URDAD analysis phase. URDAD requires that one first identifies the stakeholders which have an interest in and hence requirements for the use case/service. These are represented as interfaces.[1]

Next one identifies the pre- and post-conditions as well as quality requirements the different stakeholders have for the

[1] URDAD uses interfaces for roles because they conceptually represent roles. Actors are problematic as the object may or may not be external depending on the level of granularity and that actors cannot absorb any service requirements. For example, a user may have to do certain things when using a service. Using interfaces enables us to absorb these service requirements for external objects in a contract for that role.

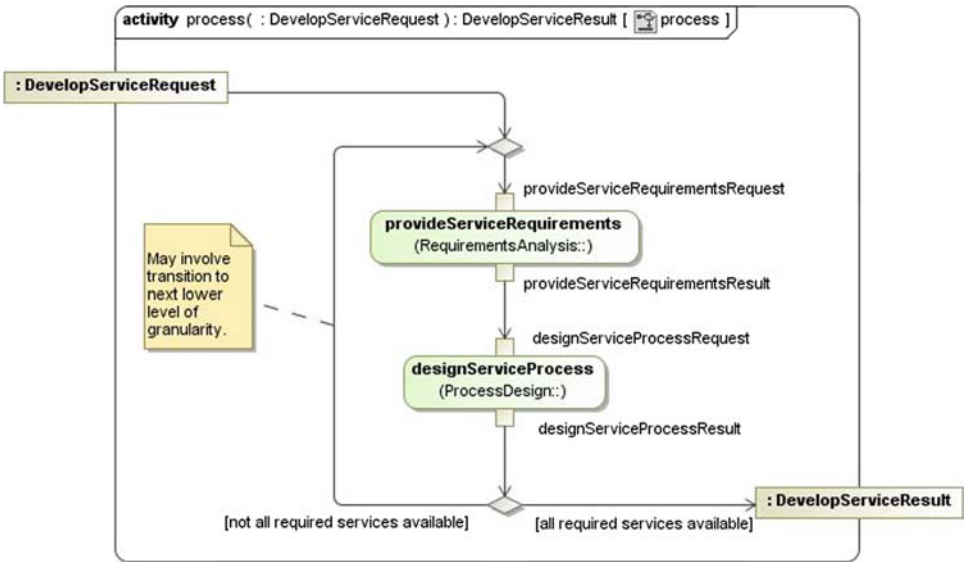

Fig. 1 High-level view of URDAD showing analysis and design across levels of granularity

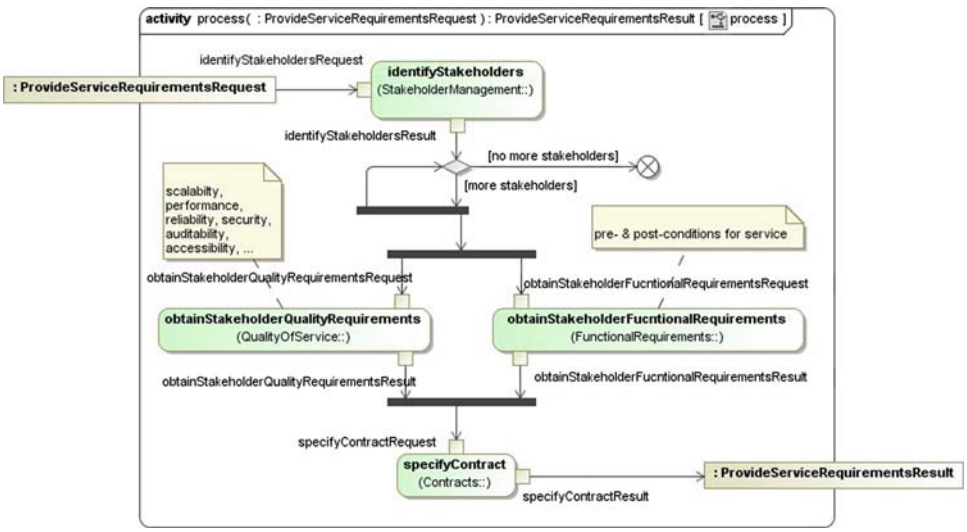

Fig. 2 The steps for the analysis phase of URDAD

service. As these are independent, this can be done concurrently. The aggregated service requirements for the current level of granularity are ultimately encapsulated in a contract for the service which is visually represented in UML as a class diagram. Initially, the domain experts may specify the pre- and post-conditions and quality requirements in English. These are incrementally formalized by mapping the pre- and post-conditions onto OCL and the quality requirements onto the UML Quality of Service (QOS) profile [22]. A simplified example of such a class diagram is shown in Fig. 3.

Note that URDAD requires the linkage between stakeholders and their functional and non-functional requirements via $<<requires>>$ dependencies in the UML model maintaining the information on who requires what.

URDAD also requires that there is a single request and a single result object for each service. This improves understandability and maintainability, avoiding scenarios where change requests result in direct interface changes and potentially in long, sparsely populated parameter lists with different scenarios requiring different sets of parameters to be provided. The data structure for the result object is populated from the user requirements. The data structure for the request object is, however, initially left blank and is incrementally populated as one does the design across levels of granularity and identifies the need for certain information in order to render the required service.

2.3 The design phase

The steps for the design phase of URDAD are shown in Fig. 1. One first identifies the services required for the enforcement of each pre- and post-condition. Next one tries

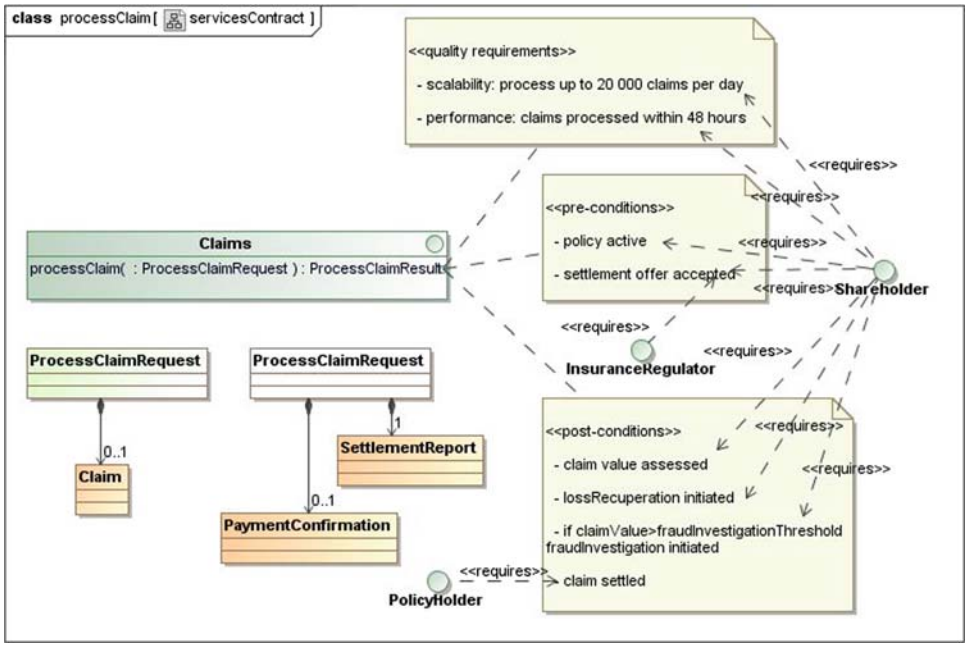

Fig. 3 A simplified services contract for a process claim use case

to group services into cohesive higher level services. This is required to fix the level of granularity in a repeatable way. One then assigns services to separate services contracts. Finally, one assembles the process realizing the service from these lower level services (Fig. 4).

The URDAD model can be populated through the use of two diagrams, a class diagram for the required services and the services contracts to which these services are assigned and an activity diagram for the process specification (Fig. 5).

The service requirements diagram shows the services used to realize the various pre- and post-condition as well as the services contracts to which these services have been assigned.

The process specification shows how the process is assembled across services sourced from the various service providers (Fig. 6).

2.4 UML concerns

URDAD requires the linkage between stakeholders and their requirements (pre- and post-conditions and quality requirements) via $<< requires >>$ dependency defined in the URDAD profile as well as the linkage between pre- and post-conditions and the services required to address them via standard UML $<< uses >>$ dependencies. Whilst this is all fine within the UML model, the UML specification does not specify any standard notation for rendering constraints like pre- and post-conditions. The support for rendering constraints is thus tool dependent and may be non-existent. This deficiency may require domain specialists to insert the dependency relationships at the model level.

3 Semi-formal elements of URDAD

The two aspects of URDAD introduced enforce a level of formality on the model. First, it is the requirement that the requirements for any service are specified as a formal services contract from which tests which assess whether any designed or implemented service fulfills the contract. The second aspect of URDAD is the formal definition of the URDAD model structure and content which is verifiable through a model validation suite.

3.1 Formal contracts-based approach

Since the model should preferably be developed in the problem/requirements domain and not in the technical/solution domain, an axiomatic/contracts-based approach is natural to model-driven development [3]. URDAD uses this formal contracts-based approach, facilitating the auto-generation of algorithm-independent tests [13]. This is 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.

3.2 Well-defined, verifiable model structure

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 validates that the UML model complies to the constraints of an URDAD metamodel, and

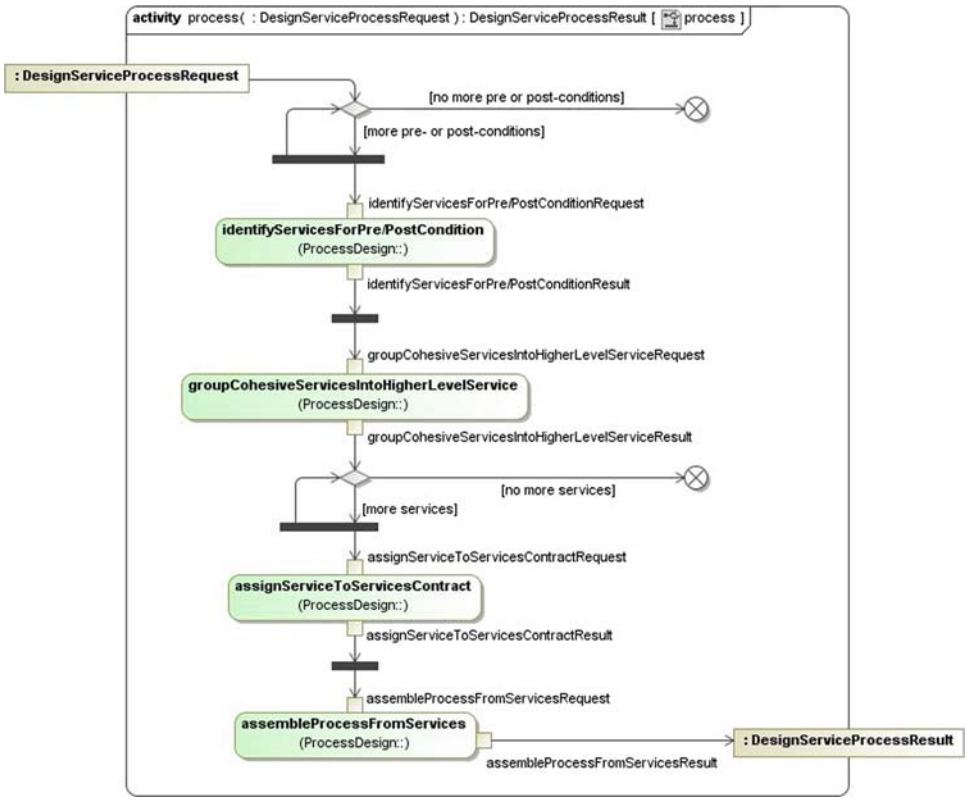

Fig. 4 The steps for the design phase of URDAD

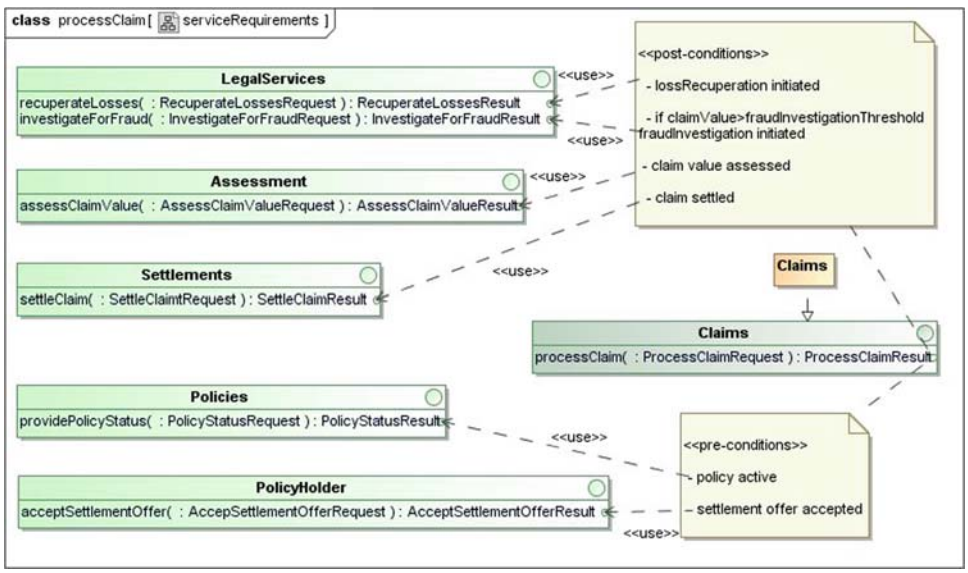

Fig. 5 The service requirements and responsibility allocation step

– 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-conditions are 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,

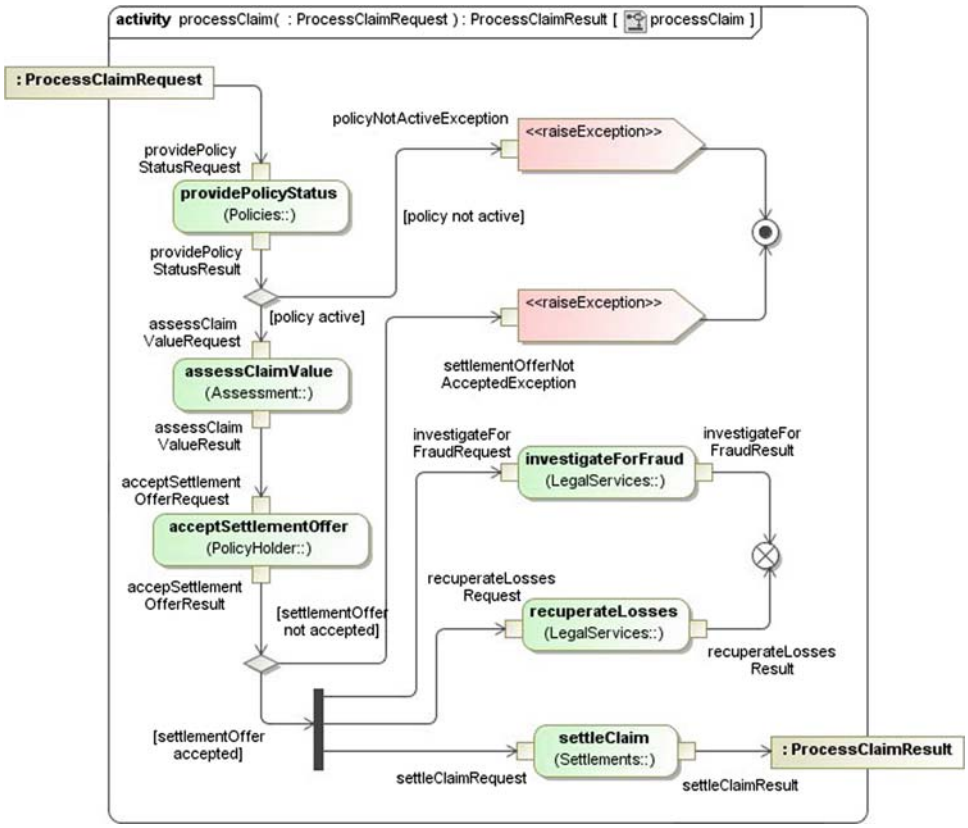

Fig. 6 Process specification for process claim service

- 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.

3.3 Incremental formalization of model

The requirements specification in the form of formal contracts using the OCL is, however, non-trivial. It is generally not feasible to expect domain specialists like business analysts to specify formal services contracts. URDAD allows for domain experts to specify services contracts informally (using natural language to specify constraints, for example), with OCL experts subsequently mapping such statements onto OCL.

In order to automate the full code generation, OCL experts will have to formalize the following aspects of the design:

- One needs to specify in OCL constraints how request objects are formally constructed from the available information.
- The formal specification of guard conditions in OCL.

4 What makes an URDAD-based approach agile?

Agile approaches [2,11] 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.

4.1 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 [7], 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.

4.2 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.
- rapid informal contract and design specification by domain specialists followed by incremental formalization of the design by technical specialists.

5 Conclusions and outlook

Over the years URDAD has strengthened its formal aspects through the introduction of model validation and formal model structure specification as well as the use of OCL to formalize the contracts specification. This has simplified model testing and 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.

References

1. Beck K, Andres C (2004) Extreme programming explained, 2nd edn. Addison-Wesley, Reading
2. Beck K, Fowler M, Martin RC et al (2001) The agile manifesto. http://agilemanifesto.org/
3. Briand LC, Labiche Y, Di Penta M, Yan-Bondoc H (2005) An experimental investigation of formality in uml-based development. IEEE Trans Softw Eng 31(10):833–849
4. Bruel J-M, Cheng B, Easterbrook S, France R, Rumpe B (1998) Integrating formal and informal specification techniques. why? how? In: 2nd IEEE workshop on industrial strength formal specification techniques, 1998. Proceedings, pp 50–57
5. Ghosh S, Dinh-Trong T, France RB, Solberg A (2006) Model-driven development using UML-2: promises and pitfalls. Computer 39(2):59–66
6. Hall A, Isaac D (1992) Software in air traffic control systems. In: IEE colloquium on the future, Jun 1992, pp 7/1–7/4
7. Kirshin A, Dotan D, Hartman A (2007) A UML simulator based on a generic model execution engine. In: Lecture notes in computer science, vol 4364/2007. Springer, Berlin, pp 324–6
8. Klopper R, Gruner S, Kourie D (2007) 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. ACM international conference proceeding series, vol 226. ACM Press, New York, pp 56–65
9. Lazar I, Parv B, Motogna S, Czibula I-G, Lazar C-L (2008) 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. IEEE Computer Society, Washington, DC, pp 38–44
10. Ma Z, He X, Kang L (2009) A model driven development platform for service-oriented applications. In: World conference on services—II, SERVICES-2 '09, pp 17–24
11. Martin RC (2002) Agile software development, principles, patterns, and practices. Prentice-Hall, Englewood Cliffs
12. McCumber WE, Cheng BHC (2001) A general framework for formalizing UML with formal languages. In: Proceedings of the 23rd international conference on software engineering, pp 433–442
13. Meyer B, Fiva A, Ciupa I, Leitner A, Wei Y, Stapf E (2009) Programs that test themselves. Computer 42(9):46–55
14. Monin JF, Hinchey MG (2003) Understanding formal methods. Springer
15. Moosavi S, Seyyedi MA, Moghadam N (2009) A method for service oriented design. In: Sixth international conference on information technology: new generations, 2009. ITNG '09. pp 290–295
16. Oquendo F (2006) Pi-method: a model-driven formal method for architecture-centric software engineering. SIGSOFT Softw Eng Notes 31(3):1–13
17. Platzer A (2009) Verification of cyberphysical transportation systems. IEEE Intell Syst 24(4):10–13
18. Siegel J (2001) Developing in OMG's model-driven architecture. White paper, Object Management Group
19. Solms F (2007) Technology neutral business process design using URDAD. In: Fujita H, Pisanelli D (eds), New trends in software methodologies, tools and techniques, frontiers in artificial intelligence and applications. IOS Press, Amsterdam, pp 52–70
20. Solms F, Loubser D (2009) Generating mda's platform independent model using urdad. Knowl Based Syst 22:174–185
21. Stahl T, Voelter M (2004) Model-driven software development. Wiley, London
22. UML Quality of Service profile (2004) UML Profile for modeling quality of service and fault tolerance characteristics and mechanisms. document ptc/04-06-01, 2004