Submission for Transformation Tool Contest 2010

Fritz Solms and Stefan Gruner
Dept. of Comp. Science, University of Pretoria, email: fritz@solms.co.za, sgruner@cs.up.ac.za

March 21, 2010

Contents

1 Contextualization

URDAD [1] [2] [3] is a service centric analysis and design methodology which is aimed in generating MDA's PIM. An URDAD model has a defined model structure and corresponding UML/OCL encoding which is meant to simplify tasks like model transformation and model validation.

It is envisaged that the contest provides an opportunity to independently assess on the one side the suitability of an URDAD model for model transformation tasks like implementation mapping, test suite and documentation generation and, on the other side, to assess model transformation tools' ability to perform the above tasks on an URDAD compliant UML model.

2 URDAD model structure

An URDAD model is service centric. Every service is associated with a use case and a services contract. Effectively URDAD treats a use case as a diagrammatic representation of a service. Services are recursively assembled from lower level services. The leaf services are services sourced from outside the scope of the system being modeled. These services may either be low level services provided by the programming language or employed frameworks (e.g. enterprise application servers, object-relational mappers, graphic library services, ...) or services sourced from other systems or other external service providers. In either case, the service requirements for these leaf services are specified through a services contract.

2.1 Model elements for a service at a particular level of granularity

URDAD fixes levels of granularity and hence exhibits the different levels of granularity directly in the model structure. For a service/use case at any level of granularity URDAD specifies

- a services contract with the inputs and outputs, pre- and post conditions and quality requirements for the service,
- the responsibility allocation view specifying the functions required to realize the pre and post-conditions specified in the services contract, and the allocation of these services to services contracts encoded as interfaces in UML.
- the process design showing the process details of how a service is assembled from the lower level services.

2.2 UML model encoding

An URDAD model can be encoded in UML/OCL using

- a class diagram with an interface for the services contract with the pre- and post-conditions specified through OCL encoded constraints.
- class diagrams for exchanged entities/value objects,
- an activity diagram for the process specification.

In addition use case diagrams can be used to specify functional requirements and sequence diagrams to specify user work flows.

2.3 Decoupling via interfaces/contracts

URDAD enforces throughout decoupling via services contracts (encoded in UML as interfaces). In an implementation mapping it is understood that, when plugging in a particular concrete service provider to realize the services contract, an appropriate adapter is used (e.g. web services, JCA adapter, ...). If the service is to be provided by a human, a human adapter will be used (e.g. an email request channel with a link to a web-based user interface).

Actors are represented in URDAD by UML interfaces and not by UML actors. This introduces one less concept for a role and facilitates the inclusion of contractual obligations for the external objects (including the user obligations). It also avoids the problem of something being external (an actor) at one level of granularrity, but internal (not an actor) at another.

2.4 Services contract specification

A services contract is specified using a UML interface with the service, its inputs and outputs, pre- and post-conditions and quality requirements.

2.5 Process specification

In URDAD the process specification for a service is usually populated via a UML activity diagram which has a very restricted structure. The process specification is a control flow across lower level services from which the higher level service is assembled. Each activity in the process specification is thus either a call operation requesting a lower level service or a control flow activity like a decision or merge node, fork or synchronization bar. The inputs and outputs for the process are, of course, the inputs and outputs for the composite service. These are shown diagrammatically as parameter nodes.

Because the flows are control flows as executed by the control logic of the higher level service, the input and output pins are not connected via object flows as the output of one lower level service is not the input to the next lower level service. Instead, the output is received by the controller (i.e. the outer composite service) which then constructs the service request object for the next requested service in the process specification from the information which is at that stage available to the process.

Each service has one initial (entry) node which starts with the receipt of the request object and two activity final nodes, one for normal exit and one for service abortion. In the case of normal exit the service is provided and resultant return value are provided. A service can only be aborted if one of the pre-conditions for the service is not met. In such scenarios the exception associated with the pre-condition is raised and sent via a send signal action and the process flow terminates on that aborted activity final node which represents service abortion.

In a complete UML/OCL encoding of an URDAD model OCL invariance constraints are used to specify how the controller (higher level service) constructs the request object for a lower level service from the request object for the higher level service and any information so far accumulated in the process.

3 Description of proposed model

The proposed model is a model for a simplistic library system. It represents a typical simple business application with some simple business processes supported by a database.

3.1 Rationale for choosing such a model

- in its default form it is very simple, without being overly academic, in nature, i.e. it is similar in nature to many enterprise systems,
- people have generally a reasonable understanding of what would be required of a simple library system and the domain concepts are generally accessible,
- it can be easily extended into a larger model which contains more elements of typical enterprise systems like
  - different presentation layers and access channels (web, mobile devices, web services, ...),
  - it can be increased to include integration requirements (e.g. to book vendors, to other libraries, ...),
  - it can be evolved to support electronic or paper based provision of electronically sourced and generated material,
- common quality requirements like security, auditability, reliability and scalability may be applied to this model.
- it is suited to be implemented across different architectures and technologies (e.g., JavaEE, SOA, Clound computing, ...)

3.2 Scope of model

The scope of the initial model is purposefully kept very constrained. The system must support registration of library members, uptake of items into the libraries inventory and loaning of library items. An overview of the scope of the system is shown in Figure 1

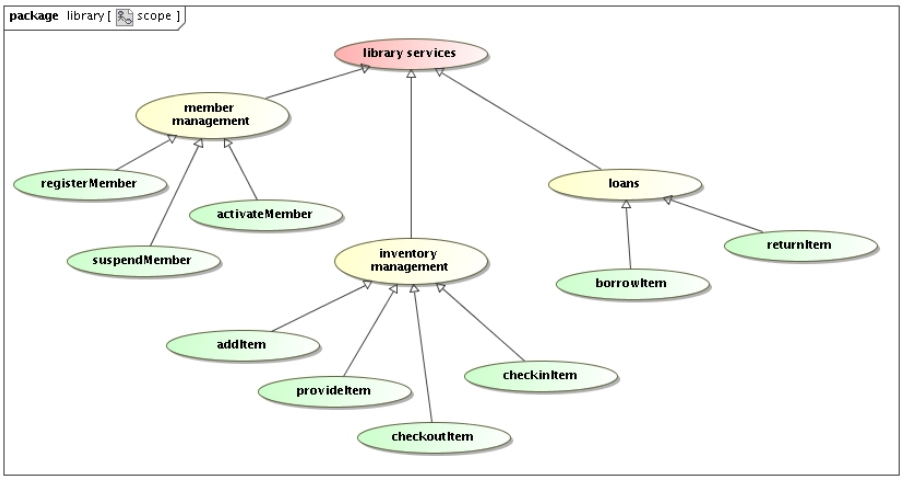

Figure 1: Scope of the library system.

4 Implementation mapping notes

This section specifies some general guidelines on the envisaged model transformations outputs of an URDAD model. The model transformation outputs include the encoding of services contracts, service implementations, unit tests for services contracts and documentation for services.

4.1 Implementation mapping of leaf services

The requirements for leaf services which are sourced from outside the scope of the system are specified through a services contract which specifies the unputs, outputs, pre- and post-conditions and quality requirements. These contracts need to be mapped onto technology representations of such contracts. Examples include WSDL, CORBA IDL and Java interface encodings.

URDAD distinguishes between errors and exceptions with errors communicating that an internal problem has occured which prevents the service provider to meet the contractual obligations for the service, while and exception communicates that a service provider does not provide the requested service because a pre-condition for the service was not met. URDAD further requires that an exception class is introduced for each pre-condition.

The model contains the linkage between a pre-condition and an associated exception class in the form of a dependency. Since UML does not specfy a notation for constraints, the dependency is generally not showable in UML diagrams. It is, however, specified in the model and URDAD supporting tools could show it diagrammatically. During implementation mapping of the services contract one needs to make certain that the correct exception is specified to be raised.

4.2 Implementation mapping of composite services

Services are recursively assembled from lower level services until a required functionality is provided by an externally available service, i.e. a service provided by the programming language, frameworks or libraries, or external systems we are integrating with.

The service contract implementation mapping is also done for non-leaf services. In addition the process realizing the service needs to be implemented.

The assembly of a process for a composite service from lower level services is specified in a UML composite activity and visually represented in a UML activity diagram. Note that the only operations contained in an URDAD compliant UML activity diagram are

- the activity for the outer service with its input and output nodes,
- call operations which request lower level services,
- send signal operations including those for raising exceptions, and
- flow control activities like decision and merge nodes, forks and sysnchronization bars.

This subset is sufficient yet minimal, has well defined semantics and allows for straight forward implementation mappings.

4.3 Implementation mapping of exchanged value objects

The URDAD model contains the data structure specifications for the exchanged data objects. These are mapped onto a technology representation like data classes in the chosen programming language, XML schemas if the encoding of the value objects is XML and so on.

When performing the model transformations for the implementation mapping of value objects one should take into account the following URDAD guidelines:

public attributes In the spirit of CORBA, the URDAD model uses public attributes on entities as a short hand notation for getter and setters.

composition implies encapsulation UML composition relationships are to be implemented in such a way that the component is encapsuled. External objects should not obtain direct access to encapsulated components. Should they require an object (e.g. an account balance) they would be given a copy of the balance object, not a direct reference or pointer to the balance object.

association The model may contain association relationships to directly accessible objects and to remote, non-accessible objects. Associations to accessible objects are implemented in a standard way via references, pointers, foreign keys etc. UML association relationships to non-accessible objects are implemented as object identifiers (e.g. URI, account number, ...).

4.4 Unit test generation

Unit tests are to be generated off service contract specifications. A unit test needs to

1. Check whether all pre-conditions are met.
2. Request the service and
   - if the pre-conditions are not met, check that an approapriate exception is thrown,
   - if the pre-conditions are met check that all post-conditions hold true after the service has been provided.

4.5 Service documentation generation

The service documentation generation is envisaged to provide a UML to natual language mapping of an URDAD/OCL model mapping the semantic statements in the model onto corresponding semantic statements in a natural language. For example, if class B is specified in the model to be a specialization of class A then this is mapped onto a statement that *instances of B are special instances of A* and so forth.

The envisaged complexity is not in the mapping of the individual semantic statements, but rather in the higher level assembly of these statements into an ordered, readable, well-structured document. For this one can be guided by the core elements of an URDAD model, i.e., that of the service contract

specification, the identification of services required to address the various pre- and post-conditions and the orchestration of those services within a defined process.

References

[1] Fritz Solms. Technology neutral business process design using urdad. In *Proceeding of the 2007 conference on New Trends in Software Methodologies, Tools and Techniques*, pages 52–70, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

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

[3] Fritz Solms and Dawid Loubser. Urdad as a semi-formal approach to analysis and design. *Innovations in Systems and Software Engineering*, 6(1-2):155–162, January 2010.