Design Management: a Collaborative Design Solution

Maged Elaasar[1] and James Conallen[2]

[1] IBM, Rational Software
770 Palladium Dr., Kanata, ON. K2V 1C8, Canada
melaasar@ca.ibm.com
[2] IBM, Rational Software
30 S 17th St # 14 Philadelphia, PA 19103, USA
jconallen@us.ibm.com

Abstract. Design is more important than ever as software systems continue to increase in complexity, become more distributed, expose multiple interfaces and have more integration points. Design process has also become more complex, involving dispersed teams, third-party components, outsourcing providers and business partners. Nevertheless, design tools have not sufficiently been coping with these growing challenges. In this paper, we discuss design challenges and highlight features of design tools that should help address them. We also describe a new application; Rational Design Management (DM) developed to boost the quality of design and streamline the design process. DM enables a collaborative approach that broadens the understanding of design, improves design quality and shrinks design time. DM leverages semantic web technologies and implements the Open Services for Lifecycle Collaboration (OSLC) specification to deliver a linked data approach for managing design. Such an approach facilitates design extensibility, reuse and integration across the development lifecycle.

Keywords: Design, Architecture, Linked Data, Design Management, UML

1 Introduction

Software design is a process of problem solving and planning for a software solution. Design has evolved from an ad-hoc, and sometimes overlooked phase, to an essential phase of any serious software development process [1]. Furthermore, the increasing complexity of today's systems has created a set of particular challenges that makes it hard for software engineers to meet the continuous customer demand for higher quality software [2]. These challenges have prompted software engineers to pay closer attention to the design process to better understand, apply, and promulgate well known design principles, processes, and professional practices to overcome these challenges. Some of the challenges include design complexity, requirements volatility, quality aspects (e.g., performance, usability, security), distributed teams, efficient allocation of resources, limited budgets, unreasonable schedules, fast-changing technology, use of third-party or open source components, and accurate transformation from software requirement to a software product.

The outcome of a design process is a software design. The impact of this design on other related software development activities cannot be underestimated. For example, good design provides an abstraction that helps determine the best possible solution. It also reduces development time through better reuse of services and less rework. In addition, it reduces integration problems through communication of and agreement on interfaces and deployment topologies. It also reduces the cost of maintenance by providing developers, who might not have been involved in the initial construction, with a blueprint of the application [3].

Furthermore, designers today vary with respect to the formality and rigor of their designs [4]. Some use informal approaches that capture design with sketches, others use more standard modeling notations (e.g., UML [5]), yet others employ full model-driven architecture (MDA) [6], which uses transformations to automate downstream activities (e.g., code generation). Also, designers differ in the nature of their development process (agile, waterfall or hybrid). Regardless of the formality or nature of the design process, designers face more or less a similar set of challenges. A lot of these challenges are a result of design tools focusing on the designer and lacking support for team aspects of software design.

Unfortunately, this lack of support for team aspects leads software designers to work in silos and be disconnected from the rest of the team [4]. Examples of this disconnection include: a) stakeholders unable to find designs related to their work and unsure if they have the latest version of the design; b) designers spending time creating static design documents to send to stakeholders; c) too much time being spent on manual design reviews late in the project or iteration only to discover changes that result in rework; d) failure to use the best people as efficiently or as broadly, as they could be; e) manually building and maintaining of spreadsheets or documents to track the impacts of requirements and design changes. We believe that a collaborative approach to design can go a long way in eliminating such scenarios. Such a collaborative approach has in fact become a necessity rather than a luxury.

In this paper, we highlight what we believe to be the most pressing software design challenges today. We also outline some of the features that software design tools must have in order to cope with those challenges. Such features are identified based on our experience developing such tools over the years and on feedback from industry practitioners. We also describe our new application in this space, called Rational Design Management (DM) [7]. DM is a server-based application that is built on the IBM Jazz platform [8]. It provides a central repository for designs and capabilities that can be accessed from either a web client or rich clients. These capabilities allow project stakeholders to easily find, access and collaborate on designs. DM also implements the emerging Open Services for Lifecycle Collaboration (OSLC) [9] specification, which is based on the principles of linked data and semantic web. This provides DM with a solid architectural foundation that facilitates design reuse, extensibility and integration with other resources across the software development lifecycle.

The rest of this paper is organized as follows: section 2 discusses the main software design challenges today and outlines a list of features that design tools should provide to address them; an overview of DM's architecture and its main features is given in section 3; section 4 elaborates on some of DM's technical design decisions that allow it to provide these features; a brief review of related tools is given in section 5; and finally section 6 provides the conclusions and highlights future works.

2 Software Design Challenges and Required Tool Features

Software designers are facing increasing business pressures to deliver faster, reduce costs and meet regulatory requirements. In trying to do so, they are confronted with challenges ([2] and [3]). We discuss here some challenges that we believe matter most to designers based on our experience helping them over the years. For each challenge, we highlight features that a modern software design tool should have to deal with it.

2.1 Expression Challenges

These challenges stem from the need to express software design using a technology that best meets the nature and goals of the design and broadens its understanding by team members. There is no doubt that the technological landscape for software design is continuously evolving and includes a myriad of formalisms; some are structured like UML and BPMN [10]; while others are less structured, like free-form sketches and rich text documents. Some of these formalisms may also need to be customized to fit the needs of a particular domains or projects. A modern software design tool should be able to not only support these formalisms, but also ensure they integrate well together to deliver synergetic value and reduce the learning curve for designers.

2.2 Access Challenges

These challenges come from the need for team members to easily find, access and collaborate on designs. When designs are hard to find or access, they tend to be built in silos, which increases the chance of discovering errors that result in rework and project cost and/or schedule overruns. They also tend to be more static and go out of sync with other related or derived resources. Moreover, they either do not get reused at all or wrong versions get used instead, resulting in a waste of design resources. They also become less comprehensible to other stakeholders like project managers, developers, testers and technical writers who have not been following them. A modern software design tool should make it easier to search for a design, access specific versions of a design, facilitate collaboration on a design and reuse of previous design components.

2.3 Lifecycle Integration Challenges

These challenges come from the need of design teams to have visibility into and coordinate activities on other interrelated lifecycle resources (e.g., requirements, work items, code, builds, test cases and test results). Faced with difficulties tracing to and assessing impact of change on those resources, designers tend to work in silos and defer coordination to the end, when rework is least possible and most expensive. This challenge occurs due to missing or poor integration between lifecycle resources that are likely to be managed by different applications. A modern software design tool should allow designs to be linked to, and have rich data-integrations with, other lifecycle resources. It should also bring transparency to the development process by enabling designers to report on, trace to and analyze how their designs relate to those resources. This would allow them to answer questions like: are all requirements covered by the design? Has the design been completely implemented? What is the status of design testing? Are quality goals being met? What needs to be changed based on changes for the design?

2.4 Awareness Challenges

These challenges relate to the need of designers to stay aware of what is going on in the project, while they are busy getting their work done, because it might impact their work or they might be able to provide valuable input. Manually searching for this information (e.g., by sending emails or attending meetings) is time consuming and is usually abandoned after project schedules start to get tight, resulting in designers becoming unaware and slipping back into their silos. A modern software design tool should allow team members to stay on top of project activities, follow project progress, see project statistics and promptly get notified of other members' requests.

2.5 Configuration Management Challenges

These challenges result from having to cope with software design variability and change. Software design may vary to cater for the needs of different users or computing environments. It also continuously changes over time to provide different or new functionality or to address problems. This motivates designers to anticipate variability and plan for it. This is also why a modern software design tool should give designers the ability to setup different but related configurations for design to reflect the varied needs of the project (e.g., different design details for different product lines, markets or computing environments). It should also allow each configuration to evolve independently over time to produce newer versions and propagate its changes to dependent configurations.

2.6 Parallel Development Challenges

These challenges appear when members of a design team work in parallel on the same design. Design teams vary in the development process they follow. One process may involve each designer working independently on a branched stream of the design and periodically merging it into an integrated stream. Such process provides the least interruption to an individual designer but makes integration between members of the design team harder. Alternatively, a process can be more agile where several designers work directly on the same copy of design. Such process leads to occasional interruption (when design components need to concurrently be changed by several designers) but provides spontaneous integration. A modern software design tool should cater to both kinds of processes. Specifically, it should ease the integration of design branches with compare and merge features. It should also improve concurrent editing experience by allowing finer componentization of design. Furthermore, it should offer design review and approval features to ease collaboration.

2.7 Summary

Software design tools must evolve to address a number of important challenges. Among them, we believe the expression challenge is the most pressing as it makes it hard to use different but related formalisms together. The next pressing challenge is lifecycle integration as it prevents linking related lifecycle artifacts making it harder to understand the complete development solution. Parallel development challenges come next, since they make a design team less productive. They are followed by configuration challenges that complicate the management of design variability. Finally, access and awareness challenges make it harder to find and reuse designs and be informed of activities on the design project.

3 Design Management: Architecture and Features

In section 2, we outlined important challenges that face software designers today. We also highlighted features that should be provided in a modern software design tool to address them. In this section, we describe the architecture of a new software design application, called Rational Design Management (DM), which we developed to boost the productivity of the design process with collaborative features. We structure the description of DM's features based on the challenges we highlighted earlier.

3.1 Architectural Overview

DM is a server-based application that adds new capabilities (e.g., a central design repository, role-based access permissions, contextual collaboration, configuration management, parallel development and lifecycle integration) to traditional software design tools, specifically Rational Software Architect (RSA) [11] and Rational Rhapsody (RHP) [12]. DM exposes these capabilities as a set of RESTful web services that are used by these tools' rich clients, as well as a web client (Figure 1).

DM allows designs to be managed in one of two modes: externally managed or actively managed. In externally managed mode, a design is managed externally to DM by another file-based software configuration management (SCM) system and periodically (e.g., daily) gets published to DM to enable other collaborative features on it (e.g., reviews, comments and links). On the other hand, in actively managed mode, a design is managed directly and exclusively by DM. One advantage of externally managed mode is the ability to access a design offline, while in actively managed mode a live connection to DM is required. Another advantage is the ability to manage both design and non-design resources, while in actively managed mode only designs are managed by DM. On the other hand, an advantage of actively managed mode is dealing with one SCM system only (i.e., DM), versus dealing with two in externally managed mode. Another advantage is an enhanced design editing experience as DM automatically manages design componentization at a more granular level than is typically done in external SCM systems, reducing the possibility of collisions or lock-outs when multiple designers work collaborative on the design.

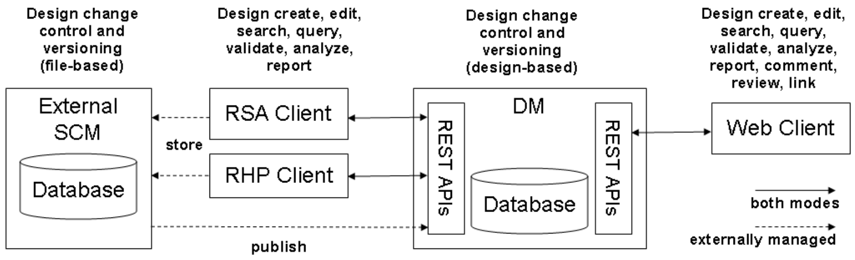

Design change control and versioning (file-based)
Design create, edit, search, query, validate, analyze, report
Design change control and versioning (design-based)
Design create, edit, search, query, validate, analyze, report, comment, review, link
External SCM
Database
RSA Client
RHP Client
store
publish
DM
REST APIs
Database
REST APIs
Web Client
both modes
externally managed

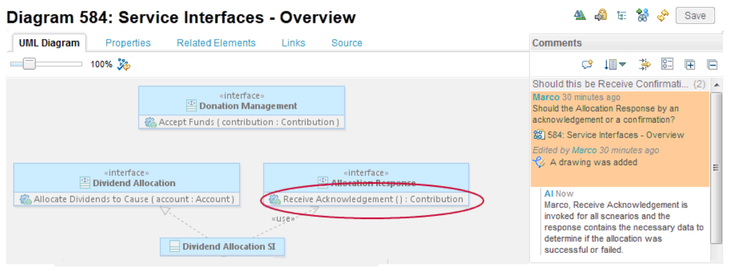
Figure 1 Design Management Architecture

3.2 Expression Features

DM supports the expression of design in various domains (formalisms). DM supports a number of structured domains (e.g., UML, BPMN and SysML [13]). It also supports less structured domains (e.g., free-form sketches and rich text documents). A design domain is defined declaratively in DM with a set of OWL [14] ontologies specifying the abstract syntax of the formalism. Designs in a given domain are represented in RDF [15] using the ontologies of the domain. In addition to the predefined domains, DM supports user-defined domains that extend predefined domains, integrate them, or define completely new ones. DM also allows the customization of domain tooling by annotating the domain's OWL ontologies with DM-specific tooling annotations.

3.3 Access Features

DM stores designs in a central repository (a RDF store) and allows role-based access (with read/write permissions) to them using a web client or rich clients. This makes it easier for designers, and other stakeholders, to find, access and browse designs. DM allows designs to be searched using keywords or queried with SPARQL [16] queries, allowing stakeholders to easily find the information they look for. It also allows them to collaborate on designs by contributing to them directly or by reviewing them with threaded comments and mark-ups (Figure 2). DM also supports design configuration management (section 3.6), which allows stakeholders to choose to access a particular version (e.g., the latest) of the design.



Should this be Receive Confirmati... (2)

Marco 30 minutes ago
Should the Allocation Response by an acknowledgement or a confirmation?
584: Service Interfaces - Overview
Edited by Marco 30 minutes ago
A drawing was added

Al Now
Marco, Receive Acknowledgement is invoked for all scnearios and the response contains the necessary data to determine if the allocation was successful or failed.

Figure 2 DM screenshot showing design comments and mark-ups on a design

3.4 Lifecycle Integration Features

DM is one of several applications in an application suite called Collaborative Lifecycle Management (CLM) [17]. Other applications in the suite manage different lifecycle resources (e.g., requirements, tests, work items and builds). DM supports integration with those applications using a specification called Open Services for Lifecycle Collaboration (OSLC) [9]. By implementing OSLC, DM allows designs to have links to other lifecycle resources. A link allows navigation to the linked resource. It also allows retrieving important information (e.g., last time modified) about the linked resource that the defining application chooses to expose. DM leverages those OSLC links to generate cross-lifecycle reports. DM and other CLM applications also support the periodic publishing of parts of their resources, including OSLC links, to a common index. This allows multi-level cross-lifecycle traceability analysis with queries to this index (Figure 3). This analysis can answer specific questions on the relationship between designs and other linked resources.

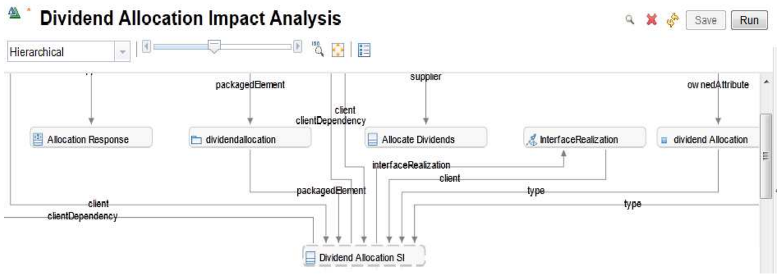
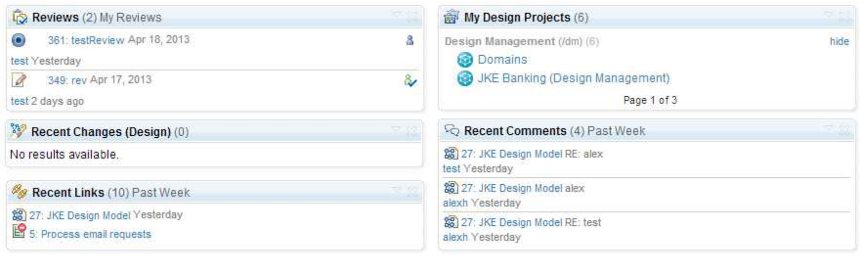


Figure 3 DM screenshot showing impact of a change in Dividend Allocation

3.5 Awareness Features

DM provides a web-based dashboard (e.g., Figure 4) that integrates relevant project info into a single location. The dashboard includes widgets for showing collaboration details, lifecycle traceability links and design queries. For example, dashboard widgets can show users which design resources have the most comments over the past week. A dashboard is not just limited to designs; it can be a mashup that combines information from the entire lifecycle using OSLC links. It is also customizable for teams and individuals with widgets that provide news feeds, bookmarks, etc.



Reviews (2) My Reviews
361: testReview Apr 18, 2013
test Yesterday
349: rev Apr 17, 2013
test 2 days ago

Recent Changes (Design) (0)
No results available.

Recent Links (10) Past Week
27: JKE Design Model Yesterday
5: Process email requests

My Design Projects (6)
Design Management (/dm) (6) hide
Domains
JKE Banking (Design Management)
Page 1 of 3

Recent Comments (4) Past Week
27: JKE Design Model RE: alex
test Yesterday
27: JKE Design Model alex
alexh Yesterday
27: JKE Design Model RE: test
alexh Yesterday

Figure 4 Web-based dashboard screenshot showing relevant project info

3.6 Configuration Management Features

DM provides its own configuration management features. Designs in DM are organized into project areas based on the product components they belong to (e.g., for a car product, there could be project areas for engine, radiator and steering). Designs can have multiple versions, all of which are stored in the same project area. However, a project area has a specific version active at a time. This version belongs to the active configuration, which is one of many related configurations in an n-dimensional configuration space. These dimensions represent product line variability parameters (e.g., for a car product, they can be model, trim and year), but the last dimension always represents time (e.g., for a car product, it can be the milestones within a year). Related project areas (e.g., those belonging to the same product) can be associated with the same configuration space allowing them to have synchronized versions.

A configuration in DM (e.g., Figure 5 shows a web-based configuration browser) can be one of two kinds: a workspace (e.g., ChildWS) or a snapshot (e.g., SomeSS). A workspace is a mutable configuration that allows designs to be changed. This is used for active work on designs. A snapshot, on the other hand, is an immutable configuration that has frozen at a point in time. This is used for releasing milestones. A new configuration can be branched off an ancestor configuration, from which it inherits its initial content. A new workspace can be branched off an ancestor snapshot, while a new snapshot can be branched off an ancestor workspace (it is not useful to branch a snapshot from another since it cannot change).

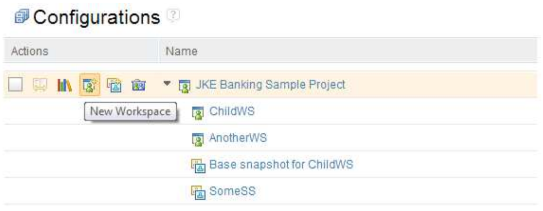


Figure 5 Configuration browser screenshot showing configuration hierarichy

3.7 Parallel Development Features

DM allows two styles of parallel development on designs. The first style is typically used in a traditional development process where each designer has a private workspace that is branched off an integration workspace (e.g., Figure 6 shows a branch workspace of project JKE Banking being edited in RSA). The designer makes changes in this branch workspace, periodically rebases by accepting latest changes from the integration workspace, and when ready delivers the branch changes to the integration workspace. This rebase and delivery operations might involve resolving conflicts when the same design components have changed. DM eases conflict resolution with a design compare and merge feature. The other style of parallel development, supported by DM, suits a more agile development process, where more than one designer works on the same active workspace in the same time. This may potentially lead to lock-outs if several designers happen to edit the same design components concurrently. To minimize this chance, DM componentizes (fragments) its designs at a more granular level. This level is configurable for each domain and is applied automatically by DM.

Furthermore, changes made by designers can be grouped into change sets (shown in Design Changes view in Figure 6). These change sets can be reverted or delivered. They can also be shared between designers for the purpose of collaboration, including commenting, reviewing and approving.

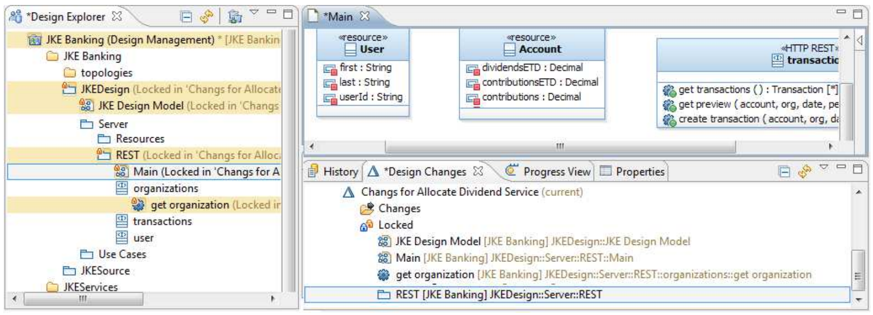

*Design Explorer
JKE Banking (Design Management) * [JKE Bankin
JKE Banking
topologies
JKEDesign (Locked in 'Changs for Allocate
JKE Design Model (Locked in 'Changs
Server
Resources
REST (Locked in 'Changs for Alloc
Main (Locked in 'Changs for A
organizations
get organization (Locked ir
transactions
user
Use Cases
JKESource
JKEServices
*Main
«resource» User
first : String
last : String
userId : String
«resource» Account
dividendsETD : Decimal
contributionsETD : Decimal
contributions : Decimal
«HTTP REST» transactic
get transactions () : Transaction [*]
get preview (account, org, date, pe
create transaction (account, org, da
History
*Design Changes
Progress View
Properties
Changs for Allocate Dividend Service (current)
Changes
Locked
JKE Design Model [JKE Banking] JKEDesign::JKE Design Model
Main [JKE Banking] JKEDesign::Server::REST::Main
get organization [JKE Banking] JKEDesign::Server::REST::organizations::get organization
REST [JKE Banking] JKEDesign::Server::REST

Figure 6 Parallel development screenshot in RSA with DM capabilities

4 Design Management: Design Decisions

In section 3, we briefly overviewed the architecture of DM and described its main features that allow it to address the design challenges outlined in section 2. In this section, we highlight some of the technical design decisions of DM that allow it to provide these features. In particular, we choose to discuss three main design decisions: 1) representing designs using RDF, 2) defining design domains using OWL, and 3) linking designs with other lifecycle resources using OSLC.

4.1 Representing Designs using RDF

Traditional software design tools, like RSA or RHP, use XSD [18] or MOF [19] based architecture to represent their designs. These architectures adopt the closed world assumption that states that what is not currently known to be true is false. This makes extending these designs, for example to add domain or project-specific extensions, quite hard. It also makes linking different designs, say a UML detailed design with a BPMN process design, a daunting task. On the other hand, the open world assumption states that the truth value of a statement is independent of whether it is known to be true by a single entity. This assumption is adopted by the semantic web [20] architecture, which represents resources in RDF as a set of RDF triples that are grouped into graphs. These graphs describe knowledge about resources, identified by their URIs, which can be extended by adding extra triples to those same graphs or to other ones.

DM chooses to represent its designs as RDF graphs. This allows designs to be easily extensible by adding extra triples. For example, one can add triples about UML notation to a UML design even though the UML ontology do not describe notation. Furthermore, this approach makes designs easily linkable to each other by adding triples representing links (cross-references). For example, one can make an activity in a UML design reference a corresponding activity in a BPMN design. Finally, a big advantage of using RDF is allowing a modular approach to design, where different aspects of a design can be specified in different RDF graphs. This also allows parallel development on designs with a reduced risk of conflicting edits. In addition, processing (e.g., querying and transforming) designs can be parallelized.

4.2 Defining Design Domains using OWL

In the semantic web architecture, knowledge about a given domain is defined as a set of ontologies. An ontology specifies, for a given domain, concepts (classes), relationships between pairs of concepts (properties), and/or individuals typed by those concepts (instances). However, an ontology is not a schema language for RDF in the traditional closed world sense. Rather, it defines the semantics of RDF data in a way that allows a reasoner to deduce new derived information. For example, knowing that John is a friend-of Bop and Bop is a friend-of Jim, and knowing that friend-of is a transitive property; a reasoner can deduce that John is also a friend-of Jim. Moreover, there is no notion of RDF graph validation in the semantic web philosophy. Rather, a reasoner can evaluate whether a set of RDF graphs are consistent, meaning they contain no contradicting statements (either directly or through inference).

The semantic web architecture defines two ontology languages: RDFS and OWL (OWL has richer semantics). DM chooses to represent the abstract syntax of its standard and proprietary design domains in OWL. However, since those standard domains are originally defined in other languages like MOF (e.g., UML), UML profile (e.g., SysML) or XSD (e.g., BPMN), DM provides mappings from those languages to OWL. For example, a MOF class, a UML stereotype or a XSD composed type maps to an OWL class. Similarly, a MOF property, a UML property or an XSD attribute/element maps to an OWL property. DM also provides reverse mappings from OWL to those languages. These mappings are used when DM designs are manipulated by rich clients of supported design tools (i.e., RSA and RHP).

Although the full details of these forward and reverse mappings are beyond the scope of this paper, we would like to elaborate on the way UML stereotypes are mapped to OWL in DM. Both a UML stereotype and its base classes in UML map to OWL classes. However, when mapping a UML element with a stereotype applied to OWL, DM represents it differently from UML. Specifically, instead of a stereotype application referencing a UML element, DM leverages RDF's multi-classification feature (the ability of a resource to have several types) to make a UML element additionally classified by the stereotype class thus has access to its properties directly.

Furthermore, DM's web-based tooling is mostly driven declaratively by domain definitions. For example, a property sheet for a design resource would list all possible properties for the resource by querying the domain's ontologies for all properties whose domain match the resource's types or one of their super types. However, DM allows customization of its web tooling through annotations to the domain's ontologies that get stored in separate graphs. For example, an annotation can override which widget to use when displaying the value of a given property in a property sheet. Such annotations are themselves defined by DM-specific tooling ontologies.

One other configuration that DM allows in a domain is design componentization. DM allows a domain to configure how to componentize a design by splitting it into different graphs. DM allows a design resource to be defined in one main graph but can be extended from other graphs. Also, DM distinguishes between a type of resource that is defined in its own graph and a type that is defined in another resource's graph. A domain can flag a class in an ontology as a graph class, which forces DM to define resources typed by this class in their own graphs. A domain can also specify for each graph class, which non-graph classes can also be defined within the same graph. For example, the UML domain in DM only flags packages, classifiers, attributes and operations as graph classes. This allows only resources of these types to have their own graphs. This automatic componentization configuration can be setup to optimize collaboration (by minimizing conflicts and concurrent edits) and linking (since only graph resources can be linked to using OSLC).

4.3 Linking Designs with Other Lifecycle Resources using OSLC

As previously mentioned in section 3.4, DM is part of the CLM suite of collaborative lifecycle applications. Lifecycle resources (e.g., requirements, designs, tests, work items) that are managed by these applications are often interrelated. These interrelationships can be specified as links between these resources. Such links allow easy navigation between the linked resources, but more importantly allow running queries, inspecting dependencies and generating reports across the lifecycle.

In order for DM (and other CLM applications) to enable this kind of linking, it implements the OSLC specification, in particular the Architectural Management (AM) sub-specification. This specification enables a linked data approach to integrating lifecycle resources. Linked data describes a method of representing resources such that they can be interlinked and be more useful. This data is often represented in RDF. In addition to the requirement that an RDF resource is identified with a web URI, a linked data approach requires such URI to be deferenceable to useful machine readable data that includes links to other related resources.

In order to implement the OSLC AM specification, DM exposes its design resources as OSLC AM resources. Specifically, it adds the oslc_am:Resource type as another type of design resources (using multi-classification). It also adds a number of expected OSLC AM properties to its resources, including a title property (`dcterms:title`) and a description property (`dcterms:description`). Since equivalent properties may already exist in a domain, DM allows a domain definition to specify those equivalent properties (e.g., a domain may specify that its `domain:name` property is equivalent to `dcterms:title`). Also, by default, DM exposes all triples of a resource to OSLC except those triples whose predicates (used properties) are flagged as private in the domain. DM uses the set of exposed (non-private) properties of a given class to construct that class's OSLC resource shape.

This shape can be retrieved for every resource and informs a linking OSLC application of what properties to expect when dealing with that resource.

5 Related Tools

While there is no disagreement on the nature of design being a collaborative effort, not all design tools today support collaborative features. This section highlights three notable design tools that boast collaborative features and compares them to DM.

Collaborative Protégé [21] is an ontology and instance editor that allows concurrent user access. It enables comments on ontologies and their changes. Users can create discussion threads, create proposals for changes, and vote on them. There is also live chat support within the editor. However, unlike DM, the tool does not have lifecycle integration capabilities, has limited SCM support, and no web access.

TopBriad Enterprise Vocabulary Net [22] is a web-based tool for the collaborative development and management of semantic web vocabulary. It supports role-based access control, vocabulary publishing, review and approval, audit-trails and parallel development. However, unlike DM, the tool has no traceability across the lifecycle and has limited SCM support.

Objecteering Teamwork [23] is a collaborative modeling tool. It supports a central shared repository, parallel development, compare/merge and integration with SCM tools. However, Unlike DM, the tool has no web-based access and no role-based user access. It also does not support user-defined domains. Finally, it does not leverage the semantic web architecture.

6 Conclusion and Future Work

The software design process has become more challenging than ever. Some of the main challenges have been outlined and discussed in this paper, including: the need to express designs in a proper formalism, the need to find, access and collaborate on designs by different stakeholders, the need to link designs to other resources in the life cycle in order to ease navigation and perform traceability analysis, the need to increase awareness of project activities that relate to design, the need to plan for design variability and change through configuration management, and the need to boost the development process by supporting parallel development.

Design Management (DM) is a new collaborative design application that we developed to address the outlined design challenges. Some of the features provided by DM include: the support of several structured and non-structured design domains with the ability to define new domains, the storage of design in a central repository with web access to ease collaboration, the ability to link designs to other lifecycle resources with the benefits of allowing traceability, analysis and report generation cross the lifecycle, the ability to have a customizable dashboard that makes designers aware of project activities, the ability for designers to create different configurations for their designs to capture their variability and perform changes in a orderly manner, the ability to comment on, review and approve designs, and finally the enablement of parallel development on designs using a traditional or an agile development process. DM is able to provide many of these features by adopting the semantic web architecture. DM can also integrate with other lifecycle applications, using a linked data approach, by implementing the OSLC specification.

Furthermore, as a new application, DM has some current limitations that we plan to overcome in future revisions. For example, it currently lacks a declarative way to define the concrete (graphical or textual) syntax of a design domain. This capability is important especially for user-defined domains. DM also lacks a way to declaratively specify a migration plan for designs when their domains evolve in a non-compatible way. Moreover, DM currently lacks a way to define inference rules in a domain to automatically compute derived information in designs to facilitate reasoning about them and checking their consistency. It also lacks a way to define and run design transformations on the server directly. We also plan to conduct case studies to evaluate DM's use in industrial settings and report on its impact on productivity.

References

1. Bass, L., Clements, P. Kazman, R.: "Software Architecture in Practice", 2nd Edition. Addison-Wesley, 2003.
2. Otero, C.: "Software Engineering Design: Theory and Practice", Auerbach Publications, June 2012.
3. Rozanski, N., Woods, E.: "Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives", 2nd Edition, 2011.
4. Leroux, D., Ramaswany, V., Yantzi, D.: "Design matters: Collaborate, automate, innovate and be agile", IBM Software Technical White paper, 2012.
5. Unified Modeling Language, Superstructure v2.2. http://www.omg.org/spec/UML/2.2/
6. Model Driven Architecture, http://en.wikipedia.org/wiki/Model-driven_architecture.
7. Rational Software Architect Design Manager: http://www-01.ibm.com/software/rational/products/swarchitect/designmanager/
8. IBM Rational Jazz Platform http://www-01.ibm.com/software/rational/jazz/
9. Open Services for Lifecycle Collaboration (OSLC) specification. http://open-services.net/
10. Business Process Model and Notation v2.0. http://www.omg.org/spec/BPMN/2.0/
11. Rational Software Architect. http://www.ibm.com/software/rational/products/ swarchitect/
12. Rational Rhapsody. http://www-142.ibm.com/software/products/us/en/ratirhaparchforsoft
13. Systems Modeling Language v1.3. http://www.omg.org/spec/SysML/1.3/
14. OWL Web Ontology Language. http://www.w3.org/TR/owl-features/
15. RDF Primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
16. SPARQL 1.1 Query Language. http://www.w3.org/TR/sparql11-query/
17. Collaborative Lifecycle Management. https://jazz.net/products/clm/
18. XML Schema. http://www.w3.org/XML/Schema.html
19. Meta Object Facility v2.0. http://www.omg.org/spec/MOF/2.0/
20. Semantic Web. http://www.w3.org/standards/semanticweb/
21. The Protégé project. http://protege.stanford.edu
22. TopBriad Enterprise Vocabulary Net. http://topquadrant.com/solutions/ent_vocab_net.html
23. Objecteering Teamwork. http://www.objecteering.com/products_teamwork.php