Modelling Support for a Linked Data Approach to Tool Interoperability

Jad El-khoury, Didem Gurdur, Frederic Loiret, Martin Törngren
Department of Machine Design
KTH Royal Institute of Technology
Stockholm, Sweden
email:{jad, dgurdur, floiret, martint}@kth.se

Da Zhang, Mattias Nyberg
Scania CV AB
Södertälje, Sweden
email: {da.zhang, mattias.nyberg}@scania.com

***Abstract*— Linked Data is increasingly being adopted for the integration of software tools, especially with the emergence of the Open Services for Lifecycle Collaboration (OSLC) standard on tool interoperability. In this paper, we present a modelling approach – with accompanying tool support – for the specification of Linked Data resources, focusing on the particular needs of tool-chain development. The approach provides graphical models for the specification of constraints on resources being shared in the tool-chain. Moreover, it aims to maintain a centralized understanding and management of the overall information model being handled in the federated tool-chain architecture. This is achieved through an integrated set of modelling views that cover the early phases of tool-chain development.**

Keywords-Linked data modelling; OSLC; resource shapes; tool integration; information modelling.

I. INTRODUCTION

Over last few decades, the ongoing trend of adopting the Model-Driven Engineering (MDE) approach to product development promised an improvement in the quality and efficient access to product and process information, given that such information becomes managed through explicitly defined meta-models. However, the heterogeneity and complexity of modern industrial products requires the use of many engineering software tools, needed by the different engineering disciplines (such as mechanical, electrical, embedded systems and software engineering), and throughout the entire development life cycle (requirements analysis, design, verification and validation, etc.).

So, while MDE is a step in the right direction, unless interoperability mechanisms are developed to connect information across the model-based engineering tools, MDE may lead to isolated "islands of information", given the natural distribution of information across the many tools and data sources involved.

As an example from the automotive industry, the functional safety standard ISO 26262:2011 [1] mandates that requirements and design components are developed at several levels of abstraction; and that a clear traceability exists between requirements from the different levels, as well as between requirements and system components. The earlier practice, in which development artefacts are handled as text-based documentation, rendered such traceability ineffective – if not impossible. Even with the adoption of model-driven engineering, it remains a challenge to trace between the artefacts being created by the various engineering tools, in order to comply with the standard.

In summary, current development practices need a faster shift from the localized document-based handling of artefacts, towards a Federated Information-based Development Environment (F-IDE), where the information from all development artefacts is made accessible, consistent and correct throughout the development phases, disciplines and tools.

In this paper, we advocate the use of the Linked Data principles as a basis for such an F-IDE (See [4] for Tim Berners-Lee's four principles of Linked Data.). Yet, when applying these principles for parts of the development environment at the truck manufacturer Scania AB, certain challenges were encountered that needed to be addressed. We here describe our approach on how these challenges were tackled. In the next section, after a short motivation for adopting the Linked Data principles, we present a case study that will be used in the remaining paper. We then further elaborate on the challenges experienced during our case study. In Section III, we describe the overall modelling approach taken to solve these challenges, followed in Section IV by detailed descriptions of the supporting models. Reflections on applying the modelling approach on the case study are then discussed in Section V.

II. PROBLEM FORMULATION

A. Background

One can avoid the need to integrate the information islands, by adopting a single platform (such as PTC Integrity [2] or MSR-Backbone [3]) through which product data is centrally managed. However, large organizations have specific development needs and approaches (processes, tools, workflow, in-house tools, etc.), which lead to a wide landscape of organization-specific and customized development environments. This landscape moreover needs to organically evolve over time, in order to adjust to future unpredictable needs of the industry. Contemporary platforms, however, offer limited customization capabilities to tailor for the organization-specific needs, requiring instead the organization to adjust itself to suite the platform. So, while they might be suitable at a smaller scale, such centralized platforms cannot scale to handle the complete

 ISBN: 978-1-61208-457-2

heterogeneous set of data sources normally found in a large organization.

A more promising approach to deal with this challenge is to adopt the concepts of Linked Data to integrate the information from the different engineering tools - without relying on a centralized integration platform. To this end, OASIS OSLC [5] is an emerging interoperability open standard that adopts the architecture of the Internet to achieve massive scalability and flexibility. OASIS OSLC is based on the W3C Linked Data initiative and follows the Representational State Transfer (REST) architectural pattern. It provides for tool- and platform-neutral usage of these web technologies to create high cohesion between tools, while reducing the need for one tool to understand the deep data of another (low coupling). This lends itself well to the distributed and organic nature of the F-IDE being desired.

When developing such a federated OSLC-based F-IDE, there is however an increased risk that one loses control over the overall product data structure that is now distributed and interrelated across the many tools. This risk is particularly aggravated if one needs to maintain changes in the F-IDE over time. In this paper, we present a modelling approach to F-IDE development that tries to deal with this risk. That is, how can a distributed architecture – as promoted by the Linked Data approach - be realized, while maintaining a somewhat centralized understanding and management of the overall information model handled within the F-IDE?

B. Case Study Description

Typical of many industrial organizations, the development environment at the truck manufacturer Scania consists of standard engineering tools, such as Jira and CAD drawing tools; as well as a range of propriety tools that cater for specific needs in the organization. Moreover, much product information is managed as generic content in office productivity tools, such as Microsoft Word and Excel.

As a subset of a larger case study, five propriety tools and data sources were to be integrated using OSLC:

1. *Code Repository* – A version-control system in which all software code resides, and from which parsers reconstruct the vehicle software architecture, based on an analysis of source code.
2. *Communication Specifier* – A tool that centrally defines the communication network of all vehicle architectures.
3. *ModArc* – A database that defines all hardware entities and their interfaces.
4. *Diagnostics Tool* - A tool that specifies the diagnostics functionality of all vehicle architectures.
5. *Requirements Specifier* - A propriety tool that allows for the semi-formal specification of system requirements.

As a first step, the data that needed to be communicated between the tools was analyzed. This was captured using a Class Diagram (Figure 1), as is the current state-of-practice at Scania for specifying a data model. For the purpose of this paper, it is not necessary to have full understanding of the data artefacts. It is worth highlighting that color-codes were used to define which tool managed which data artefact. Also, it is important to note that the model focuses on the data that needs to be communicated between the tools, and not necessarily all data available internally within each tool.

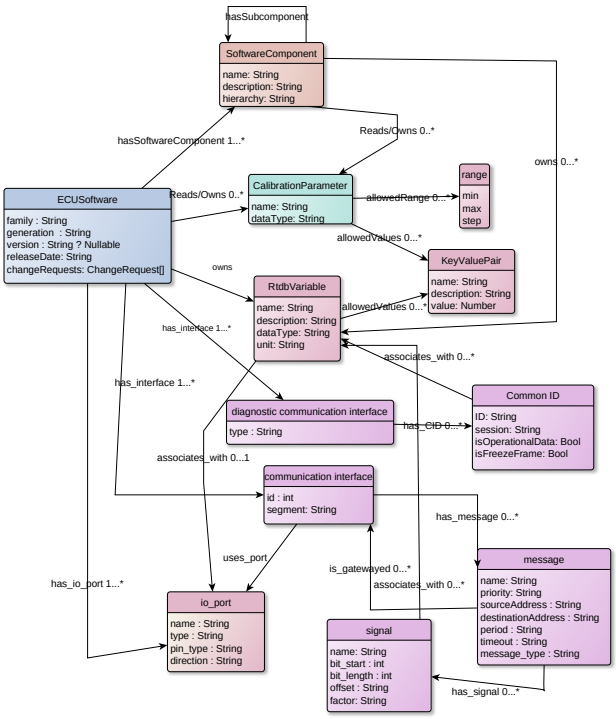

Figure 1. a UML class diagram of the resources shared in the F-IDE.

C. Identified Needs and Shortcomings

In this paper, we focus on the initial stages of specifying and architecting the desired OSLC-based F-IDE. We elaborate on the needs and shortcomings experienced by an architect during these stages:

Information specification – there is a need to specify the information to be communicated between the tools. For pragmatic reasons, a UML class diagram was adopted to define the entities being communicated and their relationships. Clearly, the created model does not comply with the semantics of the class diagram, since the entities being models are not objects in the object-oriented paradigm, but resources according to the Resource Description Framework (RDF) graph data model. Since the information model is to be maintained over time, and is also intended for communication among developers, using a class diagram - while implying another set of semantics – may lead to misunderstandings. A specification that is semantically compatible with the intended implementation technology (of Linked Data, and specifically the OSLC standard) is necessary.

Tool ownership – For any given resource being shared in the F-IDE, it is necessary to clearly identify the data source (or authoring tool) that is expected to manage that resource. That is, while representations of a resource may be freely shared between the tools, changes or creations of such a resource can only occur via its owning tool. Assuming a Linked Data approach also implies that a resource is owned

by a single source, to which other resources link. In practice, it is not uncommon for data to be duplicated in multiple sources, and hence mechanisms to synchronize data between tools are needed. For example, resources of type *Communication Interface* may be used in both *Communication Specifier* and *ModArc*, with no explicit decision on which of the tools defines it. To simplify the case study, we chose to ignore the *ModArc* source, but in reality one needs to synchronize between the two sources, as long as it is not possible to make one of them redundant.

Domain ownership – Orthogonal to tool ownership, it is also necessary to group resource definitions into domains (such as requirements engineering, software, testing, etc.). Domains can be generic in nature. Alternatively, such domain grouping can reflect the organization units that are responsible to manage specific parts of the information model. For example, the testing department may be responsible to define and maintain the testing-related resources, while the requirements department manages the definition of the requirements resources. Dependencies between the responsible departments can then be easily identified through the dependencies in the information models.

Avoid mega-meta-modelling – Information specifications originate from various development phases and/or development units in the organization. The resulting information models may well overlap, and would hence need to be harmonized. Hence, there is a need to harmonize the information models – while avoiding a central information model. Earlier attempts at information modeling normally resulted in large models that can easily become harder to maintain over time. The research project CESAR presents in [6] a typical interoperability approach in which such a large common meta-model is proposed. It is anticipated that the Linked Data approach would reduce the need to have such a single centralized mega information model. The correct handling of information through Domain and Tool Ownership (see above) ought to also help in that direction.

In summary, in architecting an F-IDE, there is a need to support the data specification using Linked Data semantics, while covering the two ownership aspects of tools (ownership from the tool deployment perspective) and domains (ownership from the organizational perspective).

III. APPROACH

We take an MDE approach to F-IDE development, in which we define models that support the architect with the needs identified in the previous section. Concretely, we present a modelling tool for the graphical definition of Linked Data resource types, based on the Linked Data constraint language of Resource Shapes [15]. Resource Shapes is a mechanism to define the constraints on RDF resources, whereby a Resource Shape defines the properties that are allowed and/or required of a type of resource; as well as each property's cardinality, range, etc.

We define the model using two views: (1) domain ownership and (2) tool ownership; with each view covering the corresponding ownership needs identified in the previous subsection.

Even though our current case study focuses on the specification and architectural design phases of F-IDE development – and in particular on information specification – we aim for an approach that can be seamlessly extended to cover the complete F-IDE life cycle, and include additional integration aspects, such as control and presentation integration [7]. Towards this, we introduce a third modelling view that supports the detailed design phase of each tool interface in the F-IDE. This view definition is made compliant with an existing code generator of tool interfaces [8]. Besides being a practical feature for the developers of tool interfaces, by ensuring that the specification model (with its three views) can lead to the generation of working code, one can validate the model's completeness and correctness with respect to the Resource Shape constraints.

The Eclipse-based modelling prototype is developed based on the Ecore meta-model of the EMF [9] project. It is important to note that adopting the close-world metamodeling approach of Ecore does not necessarily contradict the open-world view of Linked Data. The information being shared across the F-IDE remains loyal to the open-world view, within the constraints specified through the Resource Shapes mechanism. Ecore is only necessary to develop the supporting tool to define these mechanisms.

IV. THE MODEL

In this section, we present the F-IDE specification model and its three views.

Domain Specification View From this perspective, the architect defines the types of resources, their properties and relationships, using mechanisms compliant with the OSLC Core Specification [10] and the Resource Shape constraint language [15].

Figure 2 shows the Domain Specification diagram for the resources needed in our case study. The top-level container, *Domain Specification*, groups related *Resources* and *Resource Properties*. Such grouping can be associated with a common topic (such as requirements or test management), or reflects the structure of the organization managing the F-IDE. This view ought to support standard specifications, such as Friend of a Friend (FOAF) [11] and RDF Schema (RDFS) [12], as well as propriety ones. In Figure 2, three domain specifications are defined: *Software*, *Communication* and *Variability*, together with a subset of the standard domains of Dublin Core and RDF.

As required by the OSLC Core, a specification of a *Resource* type must provide a *name* and a *Type URI*. The *Resource* type can then also be associated with its allowed and/or required properties. These properties could belong to the same or any other *Domain Specification*. A *Resource Property* is in turn defined by specifying its cardinality, optionality, value-type, allowed-values, etc. Figure 3 illustrates an example property specification highlighting the available constraints that can be defined. A *Literal Property* is one whose value-type is set to one of the predefined literal types (such as string or integer); while a *Reference Property* is one whose value-type is set to either "resource" or "local resource". In the latter case, the *range* property can then be

used to suggest the set of resource types the *Property* can refer to.

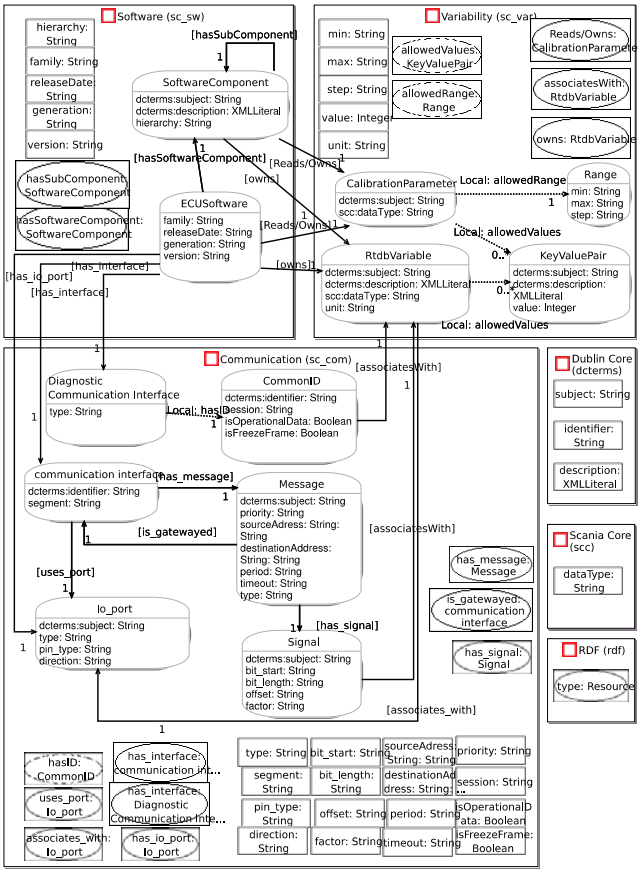

Figure 2. Domain Specification View

Borrowing from the typical notation used to represent RDF graphs, *Resource* types are represented as ellipses, while *Properties* are represented as rectangles (A *Reference Property* is represented with an ellipse within the rectangle.). In addition, *Resource Properties* are represented as first-class elements in the diagram.

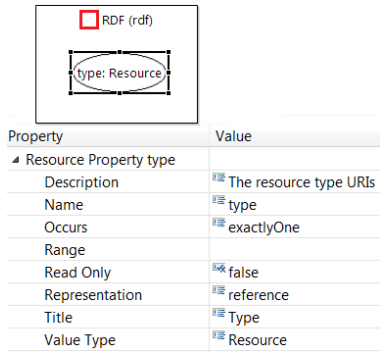

Figure 3. The specification of the rdf:type predicate, in the Domain Specification View

The association between a *Resource* type and its corresponding *Properties* is represented by arrows. However, in many cases, it becomes inconvenient to view all relationships from *Resources* to *Properties*. This is particularly the case for common *Literal Properties*, such as dcterms:subject, which can be associated to many resources across many domains. As a convenience, one can choose to hide *Resource to Literals* associations, and instead list *Literal Properties* within the *Resource* ellipse representation. It is this latter alternative that is being presented in Figure 2.

Resource Allocation View is where architect allocates resources to data sources. It gives the architect an overview of where the resources are available in the F-IDE, and where they are consumed. For each data source, the architect defines the set of resources it exposes; as well as those it consumes. These resources are graphically represented as *"provided"* (outwards arrows) and *"required"* (inwards arrows) ports on the edge of the Tool element, as illustrated in Figure 4. For example, the *Communication Specifier* tool exposes the *Message* resource, which is then consumed by the *Requirements Specifier* tool.

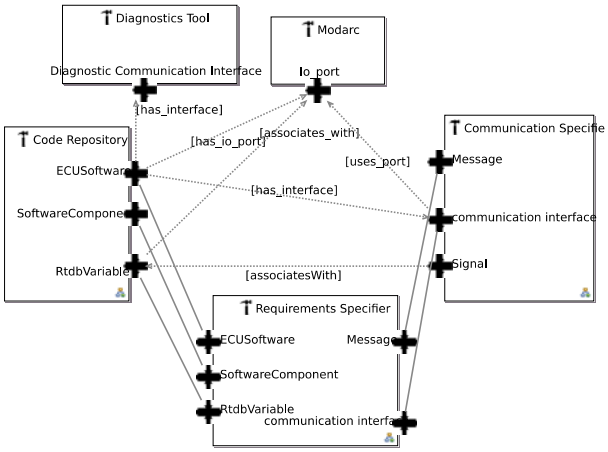

Figure 4. Resource Allocation View

In the Resource Allocation view, the interaction between a provider and consumer of a given resource is presented as a solid edge between the corresponding ports. In addition, any dependencies between resources that are managed by two different data sources are also represented in this model – as a dotted edge. For example, the resource ECUSoftware, managed by the Code Repository, has a property has_io_port that is a reference to resource IO_port; which is in turn managed through the data source Modarc. Hence, for a consumer of ECUSoftware, it is beneficial to identify the indirect dependency on the Modarc tool, since any consumption of an ECUSoftware resource, is likely to lead to the need to communicate with Modarc in order to obtain further information about the property has_io_port.

Adapter Design View is where the architect (or tool interface developer) designs the internal details of the tool interface – according to the OSLC standard. This can be performed for any of the *Tool* entities in the *Resource Allocation* view. Sufficient information is captured in this

 ISBN: 978-1-61208-457-2

view, so that an almost complete interface code, which is compliant with the OSLC4J software development kit (SDK) can be generated, based on the Lyo code generator [8].

The OSLC Core Specification defines the set of resource services that can be offered by a tool. As illustrated in Figure 5, "OSLC Services are accessible via a Service Provider that describes the Services offered. Each Service can provide Creation Factories for resource creation, Query Capabilities for resource query and Delegated UI Dialogs to enable clients to create and select resources via a web UI."[10]. The *Adaptor Design* view is a realization of the OSLC concepts in Figure 5. An example from our case study is presented in Figure 6, in which the *Core Repository* provides query capabilities and creation factories on all three resources.

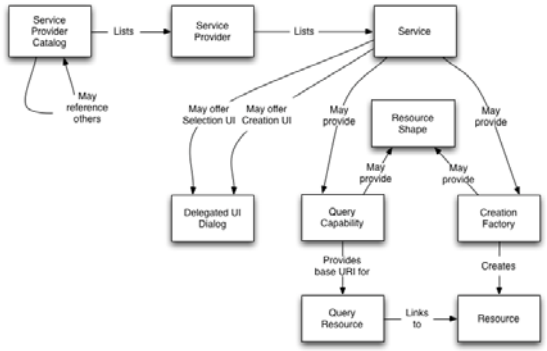

Figure 5. OSLC Core Specification concepts and relationships [10]

The *Adaptor Design* view also models its consumed resources (In Figure 6 no consumed resources are defined.). Note that the provided and required resources - as defined in this view - remain synchronized with those at the interface of the *Tool* entity in the *Resource Allocation* view.

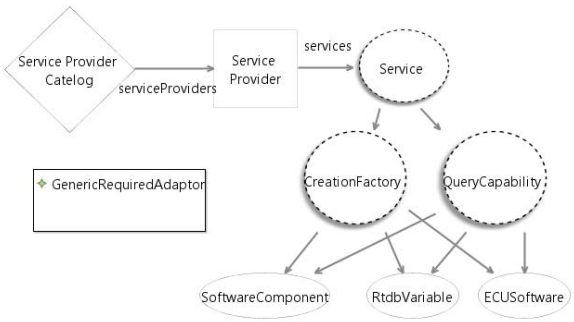

Figure 6. Adaptor Design View

There is no particular ordering of the above views, and in practice, the three views can be developed in parallel. Consistency between the views is maintained since they all refer to the same model. For example, if the architect removes a resource from the *Adaptor Design* view, the same resource is also removed from the *Resource Allocation* view.

V. REFLECTIONS

Compared to the original approach of using a UML class diagram (See Figure 1) to represent the F-IDE resources, the proposed model may seem to add a level of complexity by distributing the model information into three views. However, upon further investigation, it becomes clear that the class diagram was actually used to superimpose information for both the *Domain Specification* and *Resource Allocation* views into the same diagram. For example, classes were initially color-coded to classify them according to their owning tool. However, the semantics and intentions behind this classification soon become ambiguous, since the distinction between tool and domain ownership is not identified explicitly. In the original approach, different viewers of the same model could hence draw different conclusions when analyzing the model, depending on their implicit understanding of the color codes.

Moreover, the usage of a class diagram is not compatible with the open-world view of Linked Data, nor is it suitable to specify all necessary information according to the OSLC standard. This became apparent when the detailed specification and design of a tool's interface needed to be defined. No complements to the UML class diagram can provide such support. Instead, a dedicated domain-specific language (DSL) that follows the expected semantics can be better used uniformly across the whole organization. We here illustrate two examples where our DSL helped communicate the correct semantics, which were previously misinterpreted or not used:

- A *Resource Property* is a first-class element that can be associated with multiple *Resource* types. For example, the same allowedValues property (with range *KeyValuePair*) is a property used for both the *CalibrationParameter* & *RtdbVariable* resources. Previously, two independent properties were unnecessarily defined.
- Certain resources (such as Range) can only exist within the context of another parent resource, and hence ought not to have their own URI. Our DSL helped communicate the capability of defining *Local Resources*.

By breaking the model into two views, and by structuring each view along the managing domains and tools respectively, the information model is not expected to be developed in a top-down and centralized manner. Instead, a more distributed process is envisaged, in which resources are defined within a specific domain and/or tool. Only when necessary, such sub-models can then be integrated, avoiding the need to manage a single centralized information model.

Finally, the two orthogonal views of the F-IDE allow the architect to identify dependencies within the F-IDE, form both the organizational as well as the deployment perspective:

- In the Resource Allocation View of the model, the architect can obtain an overview of the coupling/cohesion of the tools of the F-IDE. One could directly identify the direct producer/consumer relations, as well as the indirect dependencies, as detailed in Section IV.
- In the Domain Specification view, the architect views the dependencies between the different domains (irrespective of how the resources are deployed across tools). Such dependencies reveal the relationship between the organizational entities involved in maintaining the overall information model. This explicit modelling of domain

 ISBN: 978-1-61208-457-2

ownership helps lift important organizational decisions, which otherwise remain implicit.

While the need for a dedicated DSL is convincing, the proposed views are not necessarily final, and there remains room for improvements. For example, while the possibility to represent *Properties* as first-class elements was appreciated, it was experienced that they (The squares in Figure 2) cluttered the overall model, and did not make efficient usage of the available modeling space. Similarly, the relationships between *Resources* and their associated *Literal Properties* (not shown in Figure 2) cluttered the model. Currently, filtering mechanisms are available to support different representations that suite different users, while maintaining the same underlying model. In Figure 2, the filter that hides the arrows representing relationships between *Resources* and *Literal Properties* is activated. However, the filter that hides all *Property* elements is not activated. This makes the view seem almost similar – visually - to the class diagram of Figure 1, yet the more appropriate Linked Data semantics lie behind this view.

VI. Related Work

There exists a large body of research that in various ways touches upon information modeling and model integration. (See for example [13] and [14]). Our work - and the related work of this section - is delimited to the Linked Data paradigm. The work in this paper builds upon the Resource Shape constraint language suggested in [15], by providing a graphical model to specify such constraints on RDF resources.

The most relevant work found in this area is the Ontology Definition Metamodel (ODM) [16]. ODM is an OMG specification that defines a family of Meta-Object Facility (MOF) metamodels for the modelling of ontologies. ODM also specifies a UML Profile for RDFS [12] and the Web Ontology Language (OWL) [17], which can be realized by UML-based tools, such as Enterprise Architect's ODM diagrams [18]. However, as argued in [15], OWL and RDFS are not suitable candidates to define and validate constraints, given that they are designed for another purpose - namely for reasoning engines that can infer new knowledge.

Earlier work by the authors has also resulted in a modelling approach to tool-chain development [19]. In this earlier work, even though the information was modelled targeting an OSLC implementation, the models were directly embedded in the specific tool adaptors, and no overall information model is readily available. The models did not support the tool and domain ownership perspectives identified in this paper.

VII. Conclusion

In this paper, an MDE approach to F-IDE development based on the Linked Data principles is presented. A prototype modelling tool has been developed that allows for the modelling of the information model for a complete F-IDE, based on the Resource Shapes constraint language [15]. The model is defined through three views focusing on the specification and design stages of F-IDE development. It is envisaged however that the modelling support will be extended to cover the complete development life-cycle, specifically supporting the requirements analysis phase, as well as automated testing. The current focus on data integration needs to be also extended to cover others other aspects of integration, in particular control integration [7].

The Eclipse-based prototype is to be released as an open-source contribution, yet this has not been done at the time of writing this article.

References

[1] Road vehicles - functional safety, ISO standard 26262:2011, 2011.

[2] (2015, Dec.) PTC Integrity. [Online]. Available: http://www.mks.com/platform/

[3] B. Weichel, and M. Herrmann, "A backbone in automotive software development based on XML and ASAM/MSR.", SAE Technical Papers, 2004, doi:10.4271/2004-01-0295.

[4] T. Berners-Lee. (2015, Dec.) Linked data design issues. [Online]. Available: http://www.w3.org/DesignIssues/LinkedData.html

[5] (2015, Dec.) OASIS OSLC. [Online]. Available: http://www.oasis-oslc.org/

[6] A. Rossignol, "The reference technology platform" in CESAR - Cost-efficient methods and processes for safety-relevant embedded systems, A. Rajan and T. Wahl, Eds. Dordrecht: Springer, pp. 213-236, 2012.

[7] A. I. Wasserman, "Tool integration in software engineering environments", the international workshop on environments on Software engineering environments, 1990, pp. 137-149.

[8] (2015, Dec.) Eclipse Lyo Code Generator. [Online]. Available: http://wiki.eclipse.org/Lyo/AdaptorCodeGeneratorWorkshop

[9] (2015, Dec.) Eclipse EMF. [Online]. Available: https://eclipse.org/modeling/emf/

[10] OSLC Core Specification, OSLC standard v2.0, 2013.

[11] (2015, Dec.) FOAF Vocabulary Specification. [Online]. Available: http://xmlns.com/foaf/spec/

[12] RDF Schema 1.1, W3C Recommendation, 2014.

[13] M. Törngren, A. Qamar, M. Biehl, F. Loiret, and J. El-khoury, "Integrating viewpoints in the development of mechatronic products.", Mechatronics (Oxford), vol. 24, nr. 7, 2014, pp. 745-762.

[14] R. Basole, A. Qamar, H. Park, C. Paredis, and L. Mcginnis, "Visual analytics for early-phase complex engineered system design support.", IEEE Computer Graphics and Applications, vol. 35, nr. 2, 2015, pp. 41-51.

[15] A. G. Ryman, A. Le Hors, and S. Speicher, "OSLC resource shape: A language for defining constraints on linked data.", CEUR Workshop Proceedings, Vol.996, 2013.

[16] Ontology Definition Metamodel, OMG standard, document number: formal/2014-09-02, 2014.

[17] OWL 2 Web Ontology Language, W3C Recommendation, 2012.

[18] (2015, Dec.) Enterprise Architect ODM MDG Technology. [Online]. Available: http://www.sparxsystems.com/enterprise_architect_user_guide/9.3/domain_based_models/mdg_technology_for_odm.html

[19] M. Biehl, J. El-khoury, F. Loiret, and M. Törngren, "On the modeling and generation of service-oriented tool chains.", Software & Systems Modeling, vol. 13, nr 2, 2014, pp. 461-480.

 ISBN: 978-1-61208-457-2