Towards an Ontological Model for Modelling and Automatic Code Generation of Complete Web Information Systems

Daniel Strmečki, Ivan Magdalenić
Faculty of Organisation and Informatics
Department of Information Systems Development
Pavlinska 2, 42000 Varazdin, Croatia
{daniel.strmecki,ivan.magdalenic}@foi.hr

Abstract*. The main goal of the research, which this early findings are a part of, is to enable systems development on a higher abstraction level, by applying code generation from ontological models. In this paper we present a first step towards defining an ontological model for modelling and generating complete Web information systems. The proposed model consists of several related ontologies and includes both generic and specific system requirements and functionalities. The paper includes an experiment (proof of concept), which shows that code generation from ontological models is feasible in the Web domain on a common problem case (table and input forms). Throughout the paper we present used techniques, models, technologies and frameworks.*

Keywords. Ontology, model, generator, model-driven, development, programming, automatic, generative.

1 Introduction

Code generation has been an inspiring topic for researches since the early beginnings of software engineering discipline. Due to increasing software complexity and cycles of repeating coding on different projects, we are trying to increase software reuse by raising its abstraction level. By encapsulating knowledge about lower level operations, developers can think in terms of higher level concepts and thus become more efficient (Visser, 2008). Although this topic has been researched for decades, there are no universal solutions for automatic code generation. Some of the primary reasons for this are complicated business processes, high software complexity, number of parties involved in planning and development, abilities of the creative people who are performing development activities, desire for high customisation by the client, etc. (Musen, 2000; van Ruijven, 2013)

Although we have learned from experience that we cannot automate everything, we are still trying to automate as much as we can. For example, *Software Product Lines* (SPLs) is a discipline focused on developing software families (features, variants) using specifications and code generation.

This paper focuses on code generation based on ontological models. It is a relatively new approach used in *Generative Programming* (GP) and *Model-Driven Development* (MDD) disciplines that, we believe, hasn't received enough research attention. One could argue that UML would be better suited for modelling Web information systems, or that relational databases could also be used for storing system specifications. Our literature review showed that ontological model brings several benefits, including: providing a formal language that helps bridging the gap between business and IT, enables requirements specification in an evolutionary approach, is expressive enough for specifying system features, is well suited for the definition and description of Web components, etc. (Strmečki, Magdalenić, & Kermek, 2016).

The paper presents our early findings on code generation from ontologies and a first step towards definition, standardization and reuse of an ontological model for modelling and generating complete Web information systems.

2 Background

GP can be defined as a mapping between problem space (specification) and solution space (executable code). GP applies a code generator that uses a high level specification to yield the corresponding implementation (Magdalenić, Radošević, & Orehovački, 2013). It relies on metaprogramming techniques and it is commonly applied in other disciplines like SPLs and MDD (Strmečki et al., 2016).

SPLs focus on systematically producing a set of software products consisting of a common architecture and a set of reusable assets (Asikainen, Männistö, & Soininen, 2007). Systematic, planned and strategic reuse of software assets is used to produce multiple products that satisfy a particular market segment (Duran-Limon, Garcia-Rios, Castillo-Barrera, & Capilla, 2015). Its main goal is to avoid developing software artefacts from scratch, by reconfiguring and

reusing existing products in different projects. Although each product has some specific requirements, SPLs use variability mechanisms to develop the product timely, with low cost and high quality (Nguyen, Colman, & Han, 2015).

MDD captures the essential features of a system through models and applies code generators to automatically produce the executable code from various modelled entities (Lilis, Savidis, & Valsamakis, 2014; Magdalenić et al., 2013). In MDD, models are considered to be reusable artefacts and the final model needs to be concrete enough for executable code to be generated from it (Zimmer & Rauschmayer, 2004). The archetypal MDD is based on UML, but several studies (Bartolo Espiritu, Sanchez Lopez, & Calva Rosales, 2014; Roser & Bauer, 2006; Solis, Pacheco, Najera, & Estrada, 2013; Soylu & De Causmaecker, 2009) have shown that ontological models can be efficiently used instead.

Ontologies were first introduced to software engineering trough fields of *Artificial Intelligence* and *Semantic Web*. They were soon recognized as a convenient way to describe and organize domain and software engineering knowledge. Nowadays, ontologies can be applied to aid software development in every phase of its lifecycle (Wiebe & Chan, 2012). This includes using ontologies for description of documents, formal representation of requirements, semantic description of services and components, domain modelling, executable code and test cases generation (Happel & Seedorf, 2006).

2 Related work

Since GP techniques are used in both SPLs and MDD, we will present some of the related work in those two areas. Due to paper's size limit, in this section we will shortly present only a few papers for which we think have the greatest impact on the topic.

In their approach named *Ontology Driven Architecture for Software Engineering* (ODASE), Bossche et al. presented the results from applying ontology-driven architecture on a 250 person per month e-insurance project. The focus of their research was on formalizing the requirements in order to achieve a clear contract between business and IT. They stated that ontology provides a mechanism for business to formalize their specifications and for IT to rely on the formal semantics. The knowledge transfer from ontology to executable code was done by automatic code generation based on an internal platform named Hedwig (Bossche, Ross, MacLarty, Van Nuffelen, & Pelov, 2007). ODASE showed that application of ontologies in modelling and generative software development brings several benefits, but its remains undisclosed what ontologies were used, how they were interconnected and mapped to the code generators.

Semantic Web Builder (SWB) is an agile development platform for the Web domain, which applies ontologies in requirements modelling and automatic generation of system's infrastructure. This approach generates object-oriented code from a predefined ontology that is embedded in the platform. It is a semi-automatic development platform because the resultant platform needs to be extended in order to build complex Web systems. This means that specific functionalities, like business logic and user interface, need to be developed separately and integrated with the generated code. SWB approach has been successfully applied in the development of several government Web application in Mexico (Solis et al., 2013).

Web Information System auto-construction Environment (WISE) is a prototype platform for developing Web information systems. This approach uses two embedded ontologies (domain and behaviour ontology) from which it generates executable Java code. It also provides a graphical tool called *WISE Builder* to help its users in constructing the ontologies (Tang et al., 2006). This feature can be really helpful for non-professional users, but it also makes it hard to introduce specific functionalities. Since ontologies, graphic tool and code generators are tightly coupled, we would need to extend the platform to add custom business logic or new user interface elements.

Toti and Rinelli presented a system for semi-automatic generation of an API framework on top of a semantic repository. Their system was implemented as a desktop application written in C# that provides a number of user-supervised steps for the generation process. It produces an application logic layer on top of a RDF schema which enables CRUD (create, read, update, delete) operations for each class in the RDF schema. System generates a business entity class, a DAO class, a facade class and a SOAP-based Web service interface on top of the facade for every class. In their approach, specific functionalities are not modelled trough ontologies and need to be developed separately (Toti & Rinelli, 2015).

3 Proposal

As we have shown in the short review of related work, code generation from ontological models is quite possible, although the details of its implementation are not always disclosed. Code generation from ontological models is actually a relatively popular topic in GP, SPLs and MDD disciplines nowadays. In this paper we present a proposal for an ontological model used for modelling and automatic code generation of complete Web information systems. We believe that it is possible to model complete systems through ontologies, which includes both generic and specific functionalities (like business logic and user interface).

The conceptual model on figure 1 displays multiple ontologies (and their relations) used in the proposed ontological model of Web information systems. Our proposed ontological model consists of several related ontologies (including requirements, features,

constraints, repository, forms, components and user interface) which reduces the complexity of the model and facilitates future upgrades and maintenance. In addition to serving as a specification for generating executable code, the instances of the ontological model can be viewed as a formal representation of system's features, constraints and requirements. Connections between ontological elements ensure that requirements follow the software product throughout its entire life cycle.

In a classic Web information systems development we would probably use ERA model for data representation and UML for representation of structure and behaviour. By applying the proposed ontological model instead, we can ensure preservation and possible reuse of its elements on future projects. We can achieve high reusability of the model by mapping its elements to code generators. Imagine that one already has some database table, user interface element or Web component modelled trough the ontological model and mapped to code generators. Reusing them on a new project would be as simple as specifying a pointer to the correct ontological element. The ontological element already contains all the required specification and the code generator contains the logic for producing its implementation. Of course, if one needs to develop a new component, there are some extra steps to do, like adding it to the ontological model and updating the code generators. The process of specifying and developing Web forms and their components in shown in figure 2. Even though applying an ontological model and automatic code generation creates an initial overhead in development, a high level of ontological elements reusability can bring improvements and savings in the long term. This is especially true when developing families of systems or custom made software for SMEs, with significant number of reusable artefacts.

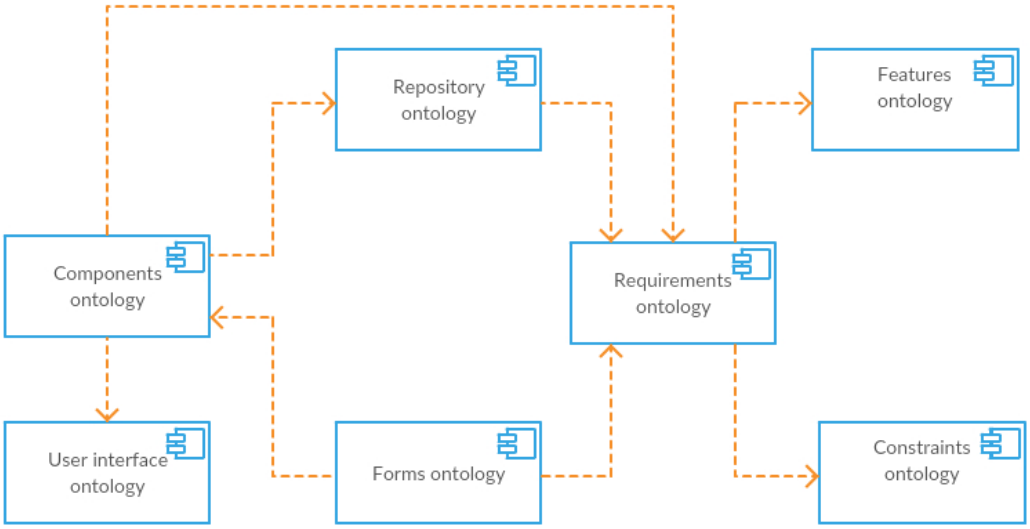

Figure 1. Ontological model – UML component diagram

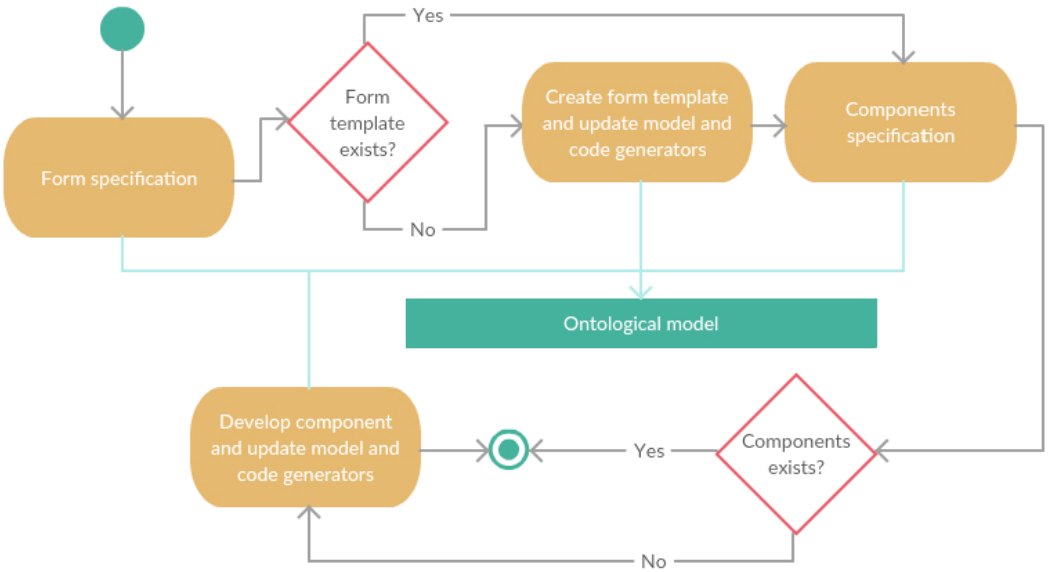

Figure 2. Process of form (and components) specification and development – UML activity diagram

4 Experiment

Our research on code generation from ontological models is still in its early stages. In this paper we will present our early findings in the form of a *Proof of Concept (*POC) application. Our POC was implemented as a Web application written in Java. It supports CRUD operations on relational database data.

4.1 Ontological model

The ontological model used in the POC shown on figure 4 is only a partial implementation of the proposed model. The focus of the POC was to demonstrate that simple, but complete Web applications can be generated from a well-defined ontological model.

The partial implementation of the proposed ontological model consist of the following ontologies: repository, forms, components and interface. The repository ontology describes the relational database domain and contains table and column definitions. The forms ontology describes the Web forms to be generated and organizes them into menu groups. The components ontology is the most complicated one, as it describes components a form can contain, as well as supported actions for clickable components. The interface ontology connects a specific component to its rendering rules (for example, the position on the page).

Ontologies were created following *Ontology 101* (Noy & McGuinness, 2001) methodology and using *Protégé 4.3.0* with *FaCT++* reasoner. The *101* methodology was selected due to its maturity, iterative nature and a large number of practical guidelines/rules for making decisions when building ontologies. *Protégé* was selected as a tool for creating ontologies because it is an open-source platform that has been proven efficient in many similar studies.

4.2 Code generators

We implemented the POC in *Java* because it a mature, high level object-oriented programming language, with good support for code templating and working with ontologies. Besides that, *Java* is one of the most popular programming languages nowadays and its code is platform independent.

The first problem one encounters when trying to generate code from an ontological model is how to access the model data from your code generators. Fortunately, there is an open source *Semantic Web* framework for *Java* we can use. It's called *Apache Jena* and among other things, it can be used to read ontological models from files or URLs and query them using *SPARQL*. *Jena* comes with a schema generator (schemagen.bat) that is used to generate a *Java* class file with all the ontological classes and properties defined within the model. Figure 5 shows an example of a *SPARQL* query with *Jena*, used to fetch all the button components for a specific Web form.

Now that we know how to read the model and its instances, we need to map the model to executable code using code generators. Code generators can be created with *Java* and *Jena* without the need for any additional frameworks and libraries. We have however decided to use an additional framework in our POC to help us with code templating. *Apache Velocity* is a templating engine for *Java* that enables developers to use a simple templating language to reference objects defined in *Java* code. Figure 6 displays an example of how we can use *Velocity* to generate getters and setters for each column in our entity *Java* class.

We used *MySQL 5.7* database and *Hibernate* object-relational mapping framework for accessing and storing data in the database. Since *Hibernate* enables high-level object handling of relational tables and columns, it makes development of code generators much easier than it would be by writing SQL statements.

Finally, the last framework we used to accelerate the development of Web forms for our POC is *Vaadin 8*. We used *Vaadin* because it provides us with a large set of free, out-of-the-box Web components that we would need to develop ourselves otherwise. It uses a component-based user interface approach for rapid building of *Java*-based Web applications, which is exactly what we need to facilitate the development of our code generators.

Having put together all the mentioned technologies and frameworks, we have successfully mapped the ontological model to executable code and generated fully functional Web forms. An example is shown in figure 3, it is a form that supports CRUD operations for a database table representing a group of articles.

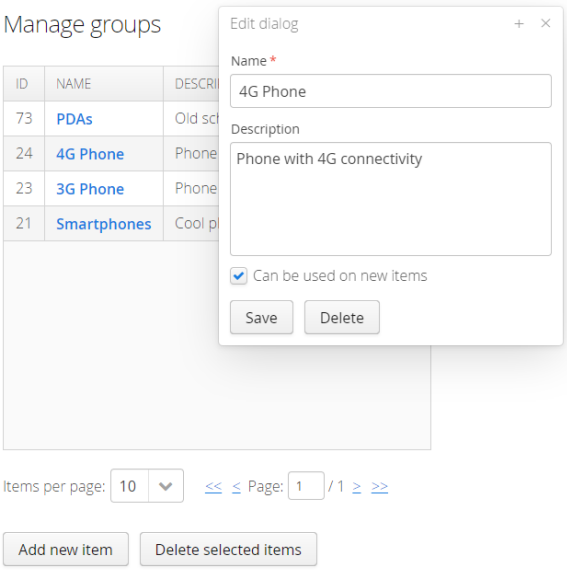

Figure 3. An example generated Vaadin Web form

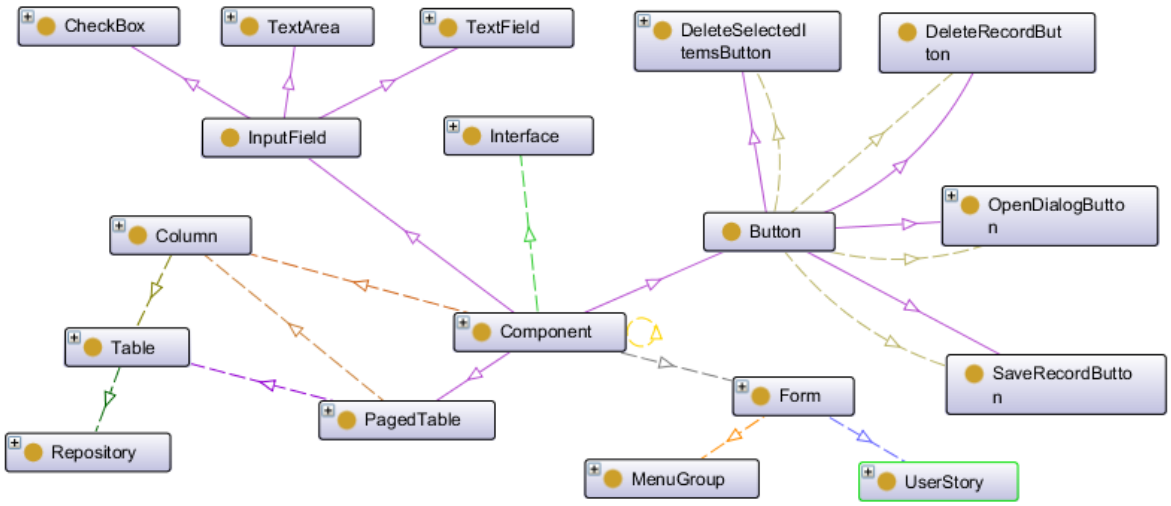

Figure 4. Ontological model used in POC – *Protégé OntoGraf*

```
StringBuilder buttonsStringBuilder = OntologyUtil.getQueryStringBuilder();
buttonsStringBuilder.append("SELECT ?resource ?type WHERE {  ");
buttonsStringBuilder.append(    "?resource rdf:type ?type . ");
buttonsStringBuilder.append(    "?resource base:componentForForm <" + formResource.getURI() + "> . ");
buttonsStringBuilder.append(    "?type rdfs:subClassOf* <" + ComponentsOntology.BUTTON + ">");
buttonsStringBuilder.append("}");
try (QueryExecution buttonsQueryExecution = QueryExecutionFactory.create(buttonsStringBuilder.toString(), model)) {
    ResultSet buttonsResultSet = buttonsQueryExecution.execSelect();
    while (buttonsResultSet.hasNext()) {
        QuerySolution buttonsQuerySolution = buttonsResultSet.next();
        Resource componentResource = buttonsQuerySolution.getResource("resource");
        Resource typeResource = buttonsQuerySolution.getResource("type");
        Button button = new Button(componentResource, typeResource);
        components.add(button);
    }
}
```

Figure 5. Querying an ontological model using SPARQL – *Apache Jena*

```
#foreach( $column in $columns )
    #if ($column.columnID != "id")
public $column.columnType get$StringUtils.capitalize($column.columnID) () {
    return $column.columnID;
}

public void set$StringUtils.capitalize($column.columnID) ($column.columnType value) {
    $column.columnID = value;
}
    #end
#end
```

Figure 6. Templating a database entity class – *Apache Velocity*

5 Conclusion

In this paper, we have presented the early findings in our research on code generation from ontological models. We presented a concept for an ontological model that can be used to model complete Web information systems, which includes both generic and specific functionalities. We reduced the complexity of the model by dividing it into several related ontologies (including requirements, features, constraints, repository, forms, components and user interface). The main goal of using an ontological model and mapping it to code generators is to achieve high reusability of modelled entities (software artefacts). We argue that in development of SPLs or custom made software for SMEs, an ontology-based generative programming approach can bring improvements and savings in the long term.

We presented an experiment, in a form of a POC, which showed that it is possible to model and generate Web forms using the proposed ontology-based generative approach. Thus, we argue that with more modelling and development effort, it is also possible to apply the same approach for developing complete Web information systems. The POC we presented relies on technologies and frameworks like *Java*, *Apache Jena*, *Apache Velocity*, *MySQL*, *Hibernate* and *Vaadin* in order to facilitate the ontological model to executable code mapping.

By applying code generation from ontological models, we are trying to enable development operations on a higher abstraction level. Ontological models helps us to move the reasoning and business logic away from hardcoded applications, which enables more efficient development and higher reusability. However, a high level of ontological knowledge reusability is required for the initial investments in ontologies and code generators to pay off. For that very reason, this paper represents a first step towards specification and standardization of an ontological model for Web information systems generative development.

6 Future work

We believe there is much more research and development to be done in this field. In our future work we will implement the complete model presented in this paper. In order to demonstrate the efficiency of this approach, we plan to carry out several case studies. Based on experience gained through all of the planned work with code generation from ontological models, we plan to define a set of guidelines for ontology-based generative development. We also plan to introduce a reasoner for our ontological model, which would help in the early detection of misspecification in the model.

References

Asikainen, T., Männistö, T., & Soininen, T. (2007). Kumbang: A domain ontology for modelling variability in software product families. *Advanced Engineering Informatics*, *21*(1), 23–40. http://doi.org/10.1016/j.aei.2006.11.007

Bartolo Espiritu, F., Sanchez Lopez, A., & Calva Rosales, L. J. (2014). Towards an improvement of software development process based on software architecture, model driven architecture and ontologies. *International Conference on Electronics, Communications and Computers*, 118–126. http://doi.org/10.1109/CONIELECOMP.2014.6808578

Bossche, M. Vanden, Ross, P., MacLarty, I., Van Nuffelen, B., & Pelov, N. (2007). Ontology driven software engineering for real life applications. *Third Int'l Workshop Semantic Web Enabled Software Eng*, 1–5. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.9906&rep=rep1&type=pdf

Duran-Limon, H. A., Garcia-Rios, C. A., Castillo-Barrera, F. E., & Capilla, R. (2015). An Ontology-Based Product Architecture Derivation Approach. *IEEE Transactions on Software Engineering*, *41*(12), 1153–1168. http://doi.org/10.1109/TSE.2015.2449854

Happel, H., & Seedorf, S. (2006). Applications of Ontologies in Software Engineering. *In 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006)*, 1–14. http://doi.org/10.1.1.89.5733

Lilis, Y., Savidis, A., & Valsamakis, Y. (2014). Staged model-driven generators: Shifting responsibility for code emission to embedded metaprograms. *MODELSWARD 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, 509–521. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84906895434&partnerID=tZOtx3y1

Magdalenić, I., Radošević, D., & Orehovački, T. (2013). Autogenerator: Generation and execution of programming code on demand. *Expert Systems with Applications*, *40*(8), 2845–2857. http://doi.org/10.1016/j.eswa.2012.12.003

Musen, M. A. (2000). Ontology-oriented design and programming. *Knowledge Engineering and Agent Technology*, 3–16. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.6062&rep=rep1&type=pdf

Nguyen, T., Colman, A., & Han, J. (2015). A Feature-Based Framework for Developing and Provisioning Customizable Web Services. *IEEE Transactions on Services Computing*, *1374*(c), 1–1. http://doi.org/10.1109/TSC.2015.2405546

Noy, N., & McGuinness, D. (2001). Ontology Development 101 : A Guide to Creating Your First Ontology. *Knowledge Systems Laboratory*.

Roser, S., & Bauer, B. (2006). Ontology-Based Model Transformation. *Satellite Events at the MoDELS 2005 Conference*, 355–356. http://doi.org/10.1007/11663430_42

Solis, J., Pacheco, H., Najera, K., & Estrada, H. (2013). A MDE Framework for semi-automatic development of web applications. *MODELSWARD 2013 - Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development*, 241–246. http://doi.org/10.5220/0004321302410246

Soylu, A., & De Causmaecker, P. (2009). Merging model driven and ontology driven system development approaches pervasive computing perspective. *2009 24th International Symposium on Computer and Information Sciences*, 730–735. http://doi.org/10.1109/ISCIS.2009.5291915

Strmečki, D., Magdalenić, I., & Kermek, D. (2016). An Overview on the use of Ontologies in Software Engineering. *Journal of Computer Science*. http://doi.org/10.3844/jcssp.2016.597.610

Tang, L., Li, H., Qiu, B., Li, M., Wang, J., Wang, L., … Tang, S. (2006). WISE: A Prototype for Ontology Driven Development of Web Information Systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 3841 LNCS, pp. 1163–1167). http://doi.org/10.1007/11610113_126

Toti, D., & Rinelli, M. (2015). Semi-automatic Generation of an Object-Oriented API Framework over Semantic Repositories. In *2015 International Conference on Intelligent Networking and Collaborative Systems* (pp. 446–449). IEEE. http://doi.org/10.1109/INCoS.2015.22

van Ruijven, L. C. (2013). Ontology for Systems Engineering. *Procedia Computer Science*, *16*, 383–392. http://doi.org/10.1016/j.procs.2013.01.040

Visser, E. (2008). WebDSL: A Case Study in Domain-Specific Language Engineering. *Generative and Transformational Techniques in Software Engineering II*, *5235*, 291–373. http://doi.org/10.1007/978-3-540-88643-3_7

Wiebe, A. J., & Chan, C. W. (2012). Ontology driven software engineering. In *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 1–4). IEEE. http://doi.org/10.1109/CCECE.2012.6334938

Zimmer, C., & Rauschmayer, A. (2004). Tuna : Ontology-Based Source Code Navigation and Annotation. *Workshop on Ontologies as Software Engineering Artifacts*, 1–9.