Extended SWRL for commercial-scale ontology-centric applications

Michel Vanden Bossche[1], Maxime Van Assche[1], Carlos Noguera[1]

[1] ODASE Ontologies SPRL, 1000 Bruxelles
{mvandenbossche,mvanassche,carlos}@odaseontologies.com

Abstract: Semantic web technologies allow a clear separation between the Business "what" and the IT "how". This is only achieved by using SWRL in addition to OWL and RDF, thereby formalizing the business logic declaratively. SWRL needs to be extended with existentials, aggregates and NAF to tackle real-world problems. This extended SWRL must leverage a syntax which is easily understood by business experts. High-performance reasoners exploiting parallelism are required to deliver a high speed of execution. By using these principles, a true continuum is established between the logic-based business specification and the technical construction relying on imperative languages used by all. This is achieved by the *ontology-centric* platform ODASE™, as proven by real-world applications in production.

Keywords: ontology, rules, OWL, SWRL, Business-IT gap, specifications, integration, commercial-scale applications, software engineering, agility, zero defects, digital transformation.

1. Introduction

The digital transformation of enterprises presents many challenges, the most critical being to develop the software that will achieve this transformation, demonstrating enterprises ability to innovate and enabling them to differentiate from their peers in an increasingly competitive environment.

Several interrelated problems restrain successful transformation. First, the increasing entropy of legacy information systems slows down their development and adds cost: a double-blow when agility and cost reduction are critical needs. Additionally, this legacy software controls critical data, a strategic asset that must be preserved. Secondly, new software developed using conventional waterfall methods is experiencing an alarmingly high failure rate (Standish, 2015), while the use of software packages can also lead to significant problems, such as Deutsche Post with DHL (Handelsblatt, 2015). Finally, the recourse to so-called "agile" methods is far from yielding convincing results (Blasband, 2016).

Of all the difficulties related to the "software crisis" (NATO, 1968), obtaining a correct and consistent specification of the business problem is the main software development obstacle: *much of the essence of building a program is in fact the debugging of the specification* (Brooks, 1987). Although less so for *S-Programs* (Lehman, 1980), those that can be derived from a formally defined specification (e.g. putting a satellite into orbit) and by formally specifiable *P-Programs* but which are not executable in-practice, except by approximations (e.g. chess), it is almost always the case with *E-Programs*, which make up the vast majority of socio-economic information systems at the heart of digital transformation.

In the latter case, creating an un-ambiguous and complete specification is a challenge rarely met. Expressed in the form of documents, it is not testable, leading to never ending discussions between Business and IT, who repeatedly fail to understand one another. IT, pressured by deadlines, advances development based on assumptions that are difficult to validate by the Business: the volume of code accumulates and invariably requires modification throughout development. These changes increase the entropy of the code and via an ever-increasing "drag effect", all the project's ambitions, including the need for agility, are eroded. So-called "agile" development suffers the same fate due to the permanent refactoring of code instituted as good practice. In this case, the root-causes of the problem are compounded given a specification based solely on rudimentary user stories.

Several approaches have been proposed to overcome these difficulties. Some are based on the preliminary definition of a model, such as MDE (Model Driven Engineering), others on the implementation of a DDD (Domain Driven Design) and a "ubiquitous language", others on the creation of DSL (Domain Specific Languages), and finally others based on Business Rules, using production rules and the pattern matching algorithm Rete.

All these approaches suffer from the same problem: they are code-centric, hence relying on a non-declarative approach (whereas the definition of the "what" must be declarative in order to be understood by the Business), without any sound theoretical basis to enhance the intrinsic software quality and to predict the consequences of design choices and inevitable changes.

Relying on a "pure" programming languages is a good option, but the use of languages such as Haskell[1] (functional) or Mercury[2] (functional and logic) remains uncommon within the developer community. The trend is against the use of such languages, as illustrated by Node.js, which makes JavaScript the current language-of-choice for "server-side" developments. But a programming language, even declarative and pure (and as such unquestionably increasing the quality of the code), remains a black-box for the business expert and therefore only marginally solves the specification problem. Finally, the accumulation of libraries and other frameworks built on languages with the greatest critical mass (Java, JavaScript) makes their use almost unavoidable for the technical construction of applications.

Whilst it is impossible to change the way programming is performed, innovation can (1) solve the specification problem and (2) create the most automated continuum possible between the Business "what" and IT "how".

2. Semantic technologies - The essential contribution of SWRL

The W3C has developed OWL and RDF as part of its Semantic Web efforts. These languages describe knowledge in a structured and meaningful way, and their primary purpose is to enable computers to conduct automated reasoning on web-based information. RDF is used for accessing data wherever it might be — in a local database or remotely somewhere on the Web — and in whatever form. OWL is a modeling language built on top of RDF.

[1] https://en.wikipedia.org/wiki/Haskell_(programming_language)
[2] https://en.wikipedia.org/wiki/Mercury_(programming_language), https://mercurylang.org/

These languages have important implications for business software development, because they allow the domain knowledge of a system to be specified completely and independently of its implementation in a way that is comprehensible by humans and computers.

In addition to OWL and RDF, semantic web technologies include SWRL, the Semantic Web Rule Language. Although a draft submission to the W3C since 2004, SWRL is actively used given its inclusion in the open source ontology editor *Protégé*[3], with a community of over 300,000 users.

SWRL is a rule language based on logic (and not on Rete-based production rules). It increases OWL expressiveness by complementing it with Horn clauses whose atoms are OWL classes and properties.

Despite the fact that the combination OWL + SWRL is undecidable, our repeated experience shows that this has no practical consequence. Conversely, the benefits of combining OWL and SWRL are significant. This is demonstrated by the many commercial-scale applications we have built since 2012, preceded by pilot developments since 1998 (with DAML until 2004).

But commercial-scale success requires practical research and development as well as sound engineering to leverage the achievements of past and ongoing academic research.

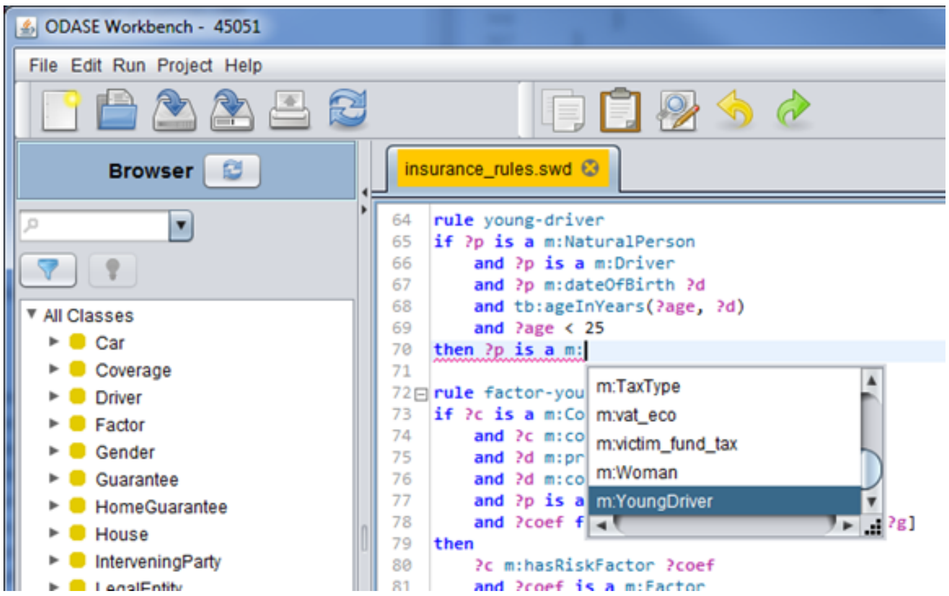
3. Extensions required by SWRL: SWORD

When we want to implement **operational** ontologies that clearly and **completely** separate the definition of the business problem from the technical implementation, the ontology must be able to specify **100%** of the business logic. We cannot therefore limit ourselves to the structural part of the ontology (OWL, RDF): the business logic needs to be part of the ontology and should not be limited to just a conceptual data model. Without SWRL, part of the business logic inevitably requires programming outside of the ontology, which would no longer guarantee a **complete** separation between the **declarative definition** of the problem and the **imperative implementation** of the solution. We then relapse into known problems associated with a partial modeling of the problem, typical of CASE, MDE and semantic web technologies limited to OWL and RDF: the business logic is programmed and no longer understandable, testable and explainable by the Business.

Our experience over the last ten years has led us (1) to extend SWRL in three directions, namely existentials, aggregates and NAF (Negation As Failure); (2) to give SWRL a textual syntax that is friendly for business experts who are not programmers; and (3) to use an ABox query-based (and not classification-based) reasoner running at high-performance such that the speed of execution of an ontology-centric application is comparable to that of conventionally-developed applications.

As an example, here is a SWORD rule defining the concept of a young driver in the case of a prototype insurance application:

[3] https://fr.wikipedia.org/wiki/Prot%C3%A9g%C3%A9_(logiciel)



```
rule young-driver
if ?p is a m:NaturalPerson
    and ?p is a m:Driver
    and ?p m:dateOfBirth ?d
    and tb:ageInYears(?age, ?d)
    and ?age < 25
then ?p is a m:

rule factor-you
if ?c is a m:Co
    and ?c m:co
    and ?d m:pr
    and ?d m:co
    and ?p is a
    and ?coef f                    ?g]
then
    ?c m:hasRiskFactor ?coef
    and ?coef is a m:Factor
```

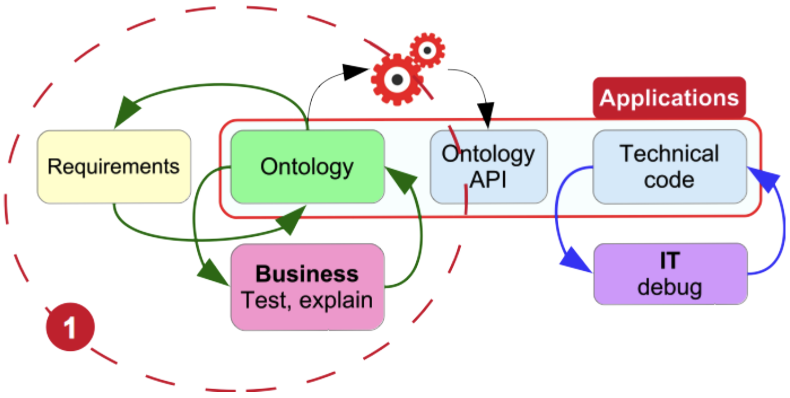
We find here the classical if … and … then structure, as well as namespaces (m = model; tb = time builtins), OWL classes (NaturalPerson, Driver, etc), properties (dateOfBirth, etc.), logical variables (?p, ?d, ?age, etc.), operators (<), etc.

4. The software development cycle with ODASE™

ODASE™ is an ontology-centric development platform and associated tools developed by Odase Ontologies (formerly Mission Critical IT) over the last ten years.

An *ontology-centric development* (OCD) is a software development approach where the ontology is a **complete and executable** model of the business problem. This model is void of any "noise" due to programming artefacts: it is purely and solely "Business", dealing with its *essential* complexity, avoiding any *accidental* complexity of the technical implementation (Brooks, 1987).

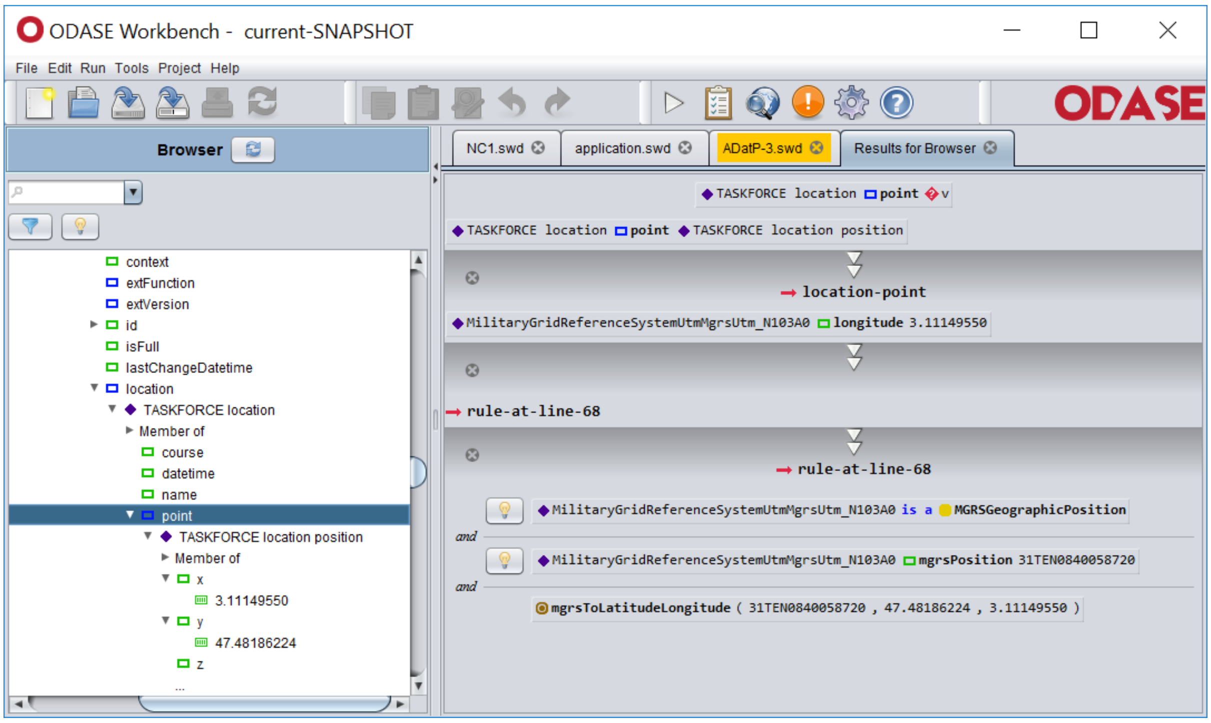
The **first step** of OCD consists of developing a business ontology (OWL, RDF) extended with SWORD rules:



The concepts (classes), attributes (datatype properties) and relationships (object properties) representing the structural component of the business are modeled using *Protégé*. SWORD rules (SWRL extended by existential, aggregates and NAF) are edited with the *ODASE*

Workbench which includes a declarative debugger explaining the results of tests, drilling down to the finest level of details.

Here is an example of an explanation[4].



ODASE Workbench - current-SNAPSHOT

File Edit Run Tools Project Help

Browser

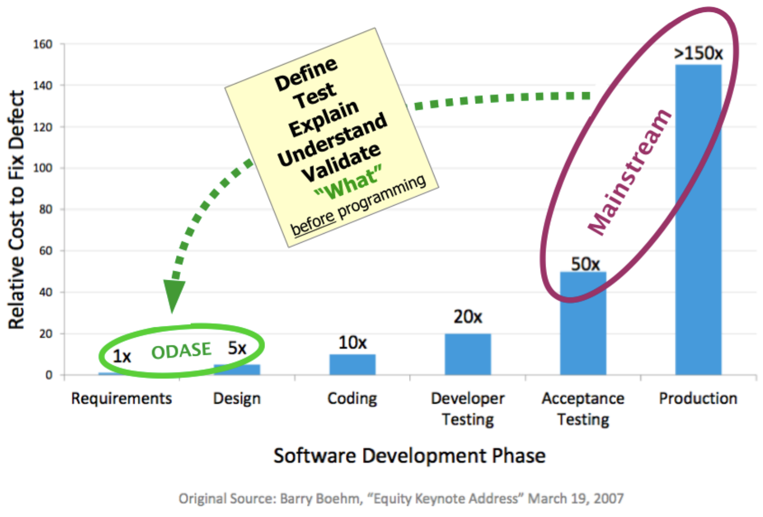
NC1.swd | application.swd | ADatP-3.swd | Results for Browser

```
context
extFunction
extVersion
▸ id
isFull
lastChangeDatetime
▾ location
  ▾ ◆ TASKFORCE location
    ▸ Member of
      course
      datetime
      name
    ▾ point
      ▾ ◆ TASKFORCE location position
        ▸ Member of
        ▾ x
            3.11149550
        ▾ y
            47.48186224
          z
          ...
```

```
◆ TASKFORCE location □ point ◆ v
◆ TASKFORCE location □ point ◆ TASKFORCE location position
→ location-point
◆ MilitaryGridReferenceSystemUtmMgrsUtm_N103A0 □ longitude 3.11149550
→ rule-at-line-68
→ rule-at-line-68
◆ MilitaryGridReferenceSystemUtmMgrsUtm_N103A0 is a ● MGRSGeographicPosition
and
◆ MilitaryGridReferenceSystemUtmMgrsUtm_N103A0 □ mgrsPosition 31TEN0840058720
and
◉ mgrsToLatitudeLongitude ( 31TEN0840058720 , 47.48186224 , 3.11149550 )
```

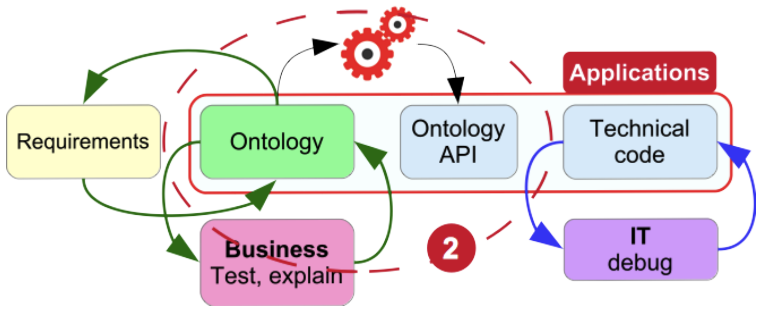
Thanks to the availability of a model with no "technical noise" and a tool explaining the results of testing, business experts can deepen their understanding of the business problem iteratively.

The specification is validated by performing tests and explaining the reason for results obtained, **before the first line of code is written**. The ontology supplemented by logic-based rules is a specification able to detect conceptual bugs at the very beginning of the development cycle when the correction of these errors is easiest and least expensive:

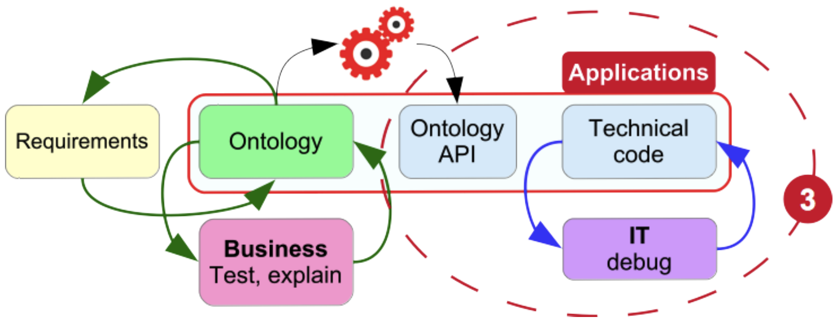


[4] Conversion of coordinates from MGRS to WGS84 in the context of messages transformation (NATO ADatP-3 to French Army NC1)

The **second step** of OCD involves the automatic generation of a type-safe API in the required language e.g. Java, C#, allowing IT to use, as-is, the ontology validated by the business:



The **third step** of OCD is where IT agilely adds the required technical code not specified in the ontology e.g. man-machine interface, database connection, middleware, etc.



Excluding these elements from the ontology enables a focus on the fundamentals of the business specification. At the same time, an operational application necessitates this code. IT no longer struggles interpreting documents, instead it receives an executable, self-documented specification validated by subject matter experts. Inferences resulting from the ontology are automatically executed “behind” the API without requiring IT intervention. Experience shows that the number of lines of programming code needed is **25 times less** compared to conventional development.

Therefore, there is a clear separation between the Business “what” and IT “how”. The gap between business and IT functions disappears since they share the same specification which is also the executable core of the application. As the ontology is validated before development, the major difficulty identified in (Brooks, 1987) is overcome.

The **fourth step** of OCD is the maintenance of the application. In the case of a business change, the ontology is first modified, tested, validated, and a new API generated. Any impact to the technical code is automatically detected during compilation: if there is no impact the server is simply restarted, alternatively the technical code is modified accordingly.

OCD is characterized by **forward** not reverse engineering: technical code is either never changed or only following a change to the ontology. We are therefore not confronted with the situation commonly associated with MDE: where it is necessary to update the model following a change during implementation.

5. Example industrial applications of ODASE™

We find in (Langevine, 2014), the presentation of an application for car insurance via multiple channels (aggregator, own website, call-centre) jointly produced by ODASE Ontologies and Aviva France, and in production since June 2014. Aviva's first challenge was to meet the need for extreme agility with no quality trade-off for the aggregator channel. This channel requires dynamic pricing such that in practice Aviva's offer is in the top three from a tariff perspective. It is also necessary to adjust the offer by adding and/or bundling options to recover margin typically eroded by the commoditization effect of direct sales. Such adjustments must be made and validated within 24 hours, not possible via COBOL given the need for a three-month lead time for new production releases. The ODASE™ application has delivered the required agility at zero defects since being put into production. Finally, it presents a performance equivalent to that of a conventional hand-written development. Aviva's Director of Information Systems observed that, compared to marketed solutions, the cost of development and maintenance has been reduced by **an order of magnitude**.

A pilot for the French Human Resources Department of the Ministry of Defense solved an integration problem between two systems: civilian human resources administration and pension management - independently developed and with a high error rate during interaction. An army expert observed: *the innovative aspect of the tool proposed consists of describing and testing operational scenarios ... The system integration benefit of the approach was not to implement an interface to generate the exchange XML flow, but to analyze business differences ensuring consistency of administrative management and pension's information. This advantage is highly valued since the guarantee that data reflects business needs cannot be given by conventional development. This is undoubtedly beneficial for a large number of projects.*

These comments illustrate the highly positive influence of the semantic expression of the business need. Another pilot concerns the correlation of railway network alarms. Several stand-alone, "siloed" information systems (e.g. interlocking, traffic management, train presence detection etc.) require semantic integration so that the data from them represents the business context. Since data are managed in real-time, it is necessary that the ontology-centric application performs at high speed, which has been proven to be the case[5].

Conclusions

The use of semantic web technologies makes it possible to effectively separate the formal definition of the problem from the technical implementation of the solution. This is only possible if SWRL rules that express business logic are included in the business ontology. SWRL as such is insufficient to model real-life applications: SWRL must be extended with existentials, aggregates and NAF. Experience shows that it is preferable to give SWRL a textual syntax that is easily understandable by the business. Finally, it is essential to use high-performance reasoners: classification-based reasoners should be avoided and the reasoners must exploit the parallelism offered by the multi-core multiprocessor systems available.

[5] See https://youtu.be/zWD8clb-eS0 for a demonstration.

By working in this way, a true continuum is achieved between the logic-based business specification and technical construct exploiting the language, libraries and frameworks on which all users depend.

The *ontology-centric* approach made possible by ODASE™ is a response to Edsger Dijkstra's 1978 observation that: *the poor quality of software is the major contributor to the soaring cost of software development*, adding: *Computing and Computing Science unavoidably emerges as an exercise in formal mathematics or, if you wish an acronym, as an exercise in VLSAL (Very Large Scale Application of Logic*[6]*).*

We think we have demonstrated that today it is not only desirable but also possible.

References

BLASBAND D. (2016). The Rise and Fall of Software Recipes. Reality Bites Publishing, Breda, Netherlands. Ch. 26 : Agile Methodologies.

BROOKS, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. In Computer, Vol. 20, No. 4 (April 1987) pp. 10-19, http://www.cs.nott.ac.uk/~pszcah/G51ISS/Documents/NoSilverBullet.html

HANDELSBLATT (2015). Post DHL Profit Plunges on Software Costs. https://global.handelsblatt.com/companies-markets/deutsche-post-dhl-profit-plunges-on-software-costs-356978

LANGEVINE L., BONE P. (2014). The Logic of Insurance: an Ontology-Centric Pricing Application. Proceedings of ISWC 2014, October 2014.

LEHMAN M. (1980). Programs, Life Cycles, and Laws of Software Evolution. Proceedings of the IEEE, Vol. 68, No 9, September 1980.

NATO (1969). Software Engineering. Peter Naur and Brian Randell., http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

STANDISH GROUP (2015). Chaos Report – Q&A with Jennifer Lynch. In InfoQ, https://www.infoq.com/articles/standish-chaos-2015

[6] https://www.cs.utexas.edu/users/EWD/transcriptions/EWD12xx/EWD1243a.html