In this thesis we study different techniques for expressing and verifying integrity constraints over data stored in ontologies built using Semantic Web technologies.

This work is motivated by data validation requirements in the @Web platform, a Semantic Web application that allows domain experts to annotate experimental data in scientific documents, and researchers to explore and query those annotations via a graphical user interface. The annotations are stored in a publicly accessible RDF graph using a vocabulary predefined in an OWL ontology, and shared with the research community.

Given the error-prone nature of the data annotation process, a set of integrity constraints has been identified that all annotated data must fulfill. It is desired to validate these constraints automatically and report any validation errors to the domain expert during the data annotation process.

To this end, we first survey the current W3C recommendations for querying, describing and constraining the contents of RDF graphs, and the available tools implementing these recommendations. We decide to focus our analysis on SPARQL, Shape Expressions and SHACL. We then implement a set of test constraints using each of the available tools and compare them according to expressiveness, verbosity, readability, etc.

Our analysis shows that none of the libraries implementing the Shape Expressions recommendation fully support all our use cases. We also observe that certain constraints expressed in SHACL require nesting SPARQL queries that are comparable in length to stand-alone SPARQL queries implementing those same constraints, thus defeating the purpose of an alternate constraint language.

We finish our analysis by comparing the running times of a set of test constraints implemented as SPARQL queries against different triple stores supported by Jena, a Java library for building Semantic Web applications which is already used in @Web for other purposes.

With this information, we choose SPARQL as the technology for expressing integrity constraints in @Web, we adopt the triple store that yields the best running times for our test constraints, and we proceed to implement the constraint verification features in the @Web platform.

To conclude, we suggest modifications to both the Shape Expressions and SHACL languages that would allow all our use cases.

**Keywords**: Semantic Web, RDF, OWL, SPARQL, Shape Expressions, SHACL, Constraints, @Web.