8. Related Work

Central to this work was the idea of showing a means of exploiting the Cloud to provide data elasticity to data-driven applications. The Container Database API is meant to be a building block of data-driven applications, providing a very simple database manager as well as a programming model where a client application has to create Entity Classes (classes that mapped to the problem domain vocabulary), associate them to local and remote storage, and finally set some rules of usage for the allocated storage.

If we were to consider other frameworks for creation of data-driven software, there are a several examples of commercial ones that provide ways to create data models and associate them to physical storage; for instance, Microsoft Entity Framework and Microsoft LightSwitch [30] both allow for defining problem domain objects, their relations, links to physical storage (including the Cloud) and views for the defined data model. However, both approaches are based on ORM, as opposed to the {Blob+Metadata} semi-structured approach proposed in this work. Most important, they are not database management systems – they are RAD (Rapid Application Development) tools that facilitate the use of specific third-party databases, like SQL Server.

In relation to the available database management systems on the market, there are hundreds of commercial options to choose, from relational DBMSs like SQL Server or MySQL to newer, no-SQL DBMSs, like the semi-structured MongoDB [31] or the graph database NEO4j [32]. As far as this work is being written, none of these systems deal with the concept of storage overflow as it is dealt in the Container Database; i.e., a system where local storage is treated as primary storage and Cloud storage is seen as overflow storage that provides data elasticity.