Flexible Data Distribution Policy Language and Gateway Architecture

Josef Spillner, Alexander Schill
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
Email: {josef.spillner,alexander.schill}@tu-dresden.de

***Abstract*—Concerns about the security and reliability of data storage in online systems, alias cloud storage services, have led to the design of application-independent storage overlays and controllers which guarantee security and optimality with regards to the requirements of their users. In roaming and multi-user environments, the configuration and rules steering these controllers should be easily extensible and migratable in order to not simply shift the vendor lock-in from the service provider to the controller software or appliance. Despite this requirement, there is no language available yet to express such rules and policies. Therefore, we introduce the Flexible Data Distribution Policy Language (FlexDDPL) to formulate storage policies, evaluate it against language design guidelines, and demonstrate its practical usability within a dispersing cloud storage gateway architecture.**

I. Introduction

Ubiquitous access to personal documents from all devices, convenient sharing of files with friends and colleagues, versioned offsite backups without own dedicated hardware, rapid access to extended capacity in moments of burst data generation - the use cases for network and online storage integration are plenty. Numerous storage service providers have emerged and gained popularity recently through the push towards cloud computing and associated cloud storage. Conscious users and researchers have nevertheless taken issue with uncontrolled replication of data into the cloud, and even more with replica-less migration, due to the reliability, security and privacy drawbacks associated with these operations. Temporary lock-out and irrevocable loss of data, expensive retrieval of accrued data sets as well as unauthorised access to and misuse of data are among the most-cited reasons for caution [1], [2].

Encryption and dispersion have been suggested as helpful techniques against these issues. As a result, cloud storage controllers have emerged as prototypical parts of operating systems and user environments [3], [4]. In single-user scenarios, the configuration is often refined to the user's desktop or personal devices. In larger groups and organisations with multiple users and departments, the controllers are instead centralised and hosted on the Intranet similar to conventional network gateways. Hence, cloud storage gateways now exist to connect large numbers of users in hierarchical organisations safely, securely and optimally to the cloud [5], [6]. An abstract view on cloud storage gateways is given in Fig. 1.

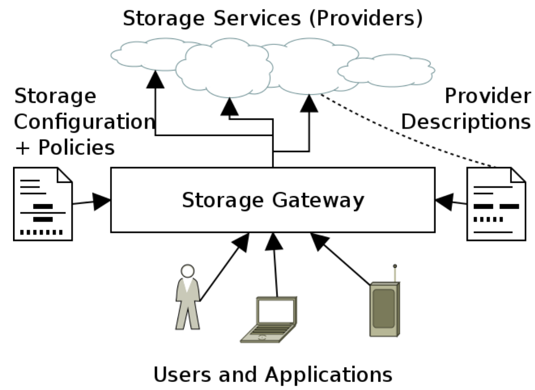

Figure 1. A cloud storage gateway and its environment

Like all enterprise integration architectures, the storage gateways are operating within a context mandated by access control restrictions, identity management providers, application integration middleware and other standard distributed system software. Along with other authorisation decisions, the use of cloud storage gateways is subject to rules and policies. To date, no domain-specific language exists yet to describe them in a sufficiently clear and portable manner.

Therefore, this article describes our contribution of a flexible data distribution policy language and its application in cloud storage gateways. The remaining sections are structured as follows: First, we consider existing policy languages in distributed systems and point out their characteristics. Then, we reason about the design and vocabulary of a flexible data distribution policy language, and specify it along with syntax samples. Subsequently, we describe the architecture and implementation of a policy-aware storage gateway. Finally, we evaluate both the language and the gateway and raise our expectations for future work.

II. Related Work

Elastic Quotas [7] allow users to specify file priorities in a multi-user shared resource environment. This declarative approach uses file and path patterns. It works well for network file servers, but is not sufficient for regulating attached remote storage services. An attempt to govern the data flows between systems is the Data Distribution Policy Expression

(DDPE) [8]. This language defines abstract high-level policies and is not suitable for fine-grained data distribution descriptions. The Policy-based Data Placement Service (PDPS) [9] is more concrete. It relies on Drools rules and can be integrated with scientific data workflow systems. However, it lacks ad-hoc service integration policies which are crucial for distributed storage in constantly changing heterogeneous environments. The Web Services Policy Language (WSPL) describes web service access and invocation policies on the basis of XACML [10]. Conceptually, the language design is suitable for multi-user storage service environments due to its design of two layers, for instance administrator and user policies, which can be merged. Similar matchmaking-enabling rule languages exist for semantic web services, for example service and goal specifications described in the Web Service Modelling Ontology (WSMO). All of these languages are restricted to the service selection phase and do not account for integration and client-side processing aspects at runtime. A proposed policy-aware cloud database management middleware, on the other hand, explicitly considers enterprise integration and multi-user policies [11]. It relies on Key-Policy Attribute Based Encryption (KP-ABE). Consequently, this approach is only applicable to outsourced databases, not to file-based storage.

Proposals for cloud storage gateways exist mostly separate from those for policy languages. A transparent compression gateway reduces the storage overhead in public infrastructure providers and speeds up data transmission between users and providers [12]. This architecture does however not include means for privacy-preserving and quality-increasing splitting of data across providers. Other gateways and controllers store their configuration scattered across configuration files (e.g. NubiSave [4]) or in files and databases (e.g. SecCSIE [5]). In these cases, it is hard to extract strategic decisions from the configuration to evolve or migrate the system.

Our contribution is therefore to introduce a portable runtime-interpreted storage policy language and gateway.

III. Data Distribution Policy Language Design

The overall goal of interpreting a document written in formal language within policy engines of data distribution architectures is to specify precisely who is allowed under what conditions to store which data how into which targets. The explicit goals of the language syntax are thus to express:

- How users or groups store data. Typically, different groups of users have different needs regarding the data transmission and storage.
- How data is divided and routed. This encompasses the static or dynamic selection of both in-house and outsourced services, and the application of redundancy and encryption.
- How data is subject to adaptation. For instance, offsite copies are stored with degraded quality for security and performance reasons.

We achieve these goals with a new domain-specific declarative language whose syntax can directly express the combination of goals as appropriately scoped rules. It is called the Flexible Data Distribution Policy Language, abbreviated as FlexDDPL.

A. FlexDDPL Syntax and Grammar

The textual syntax of FlexDDPL is line-based. Each line which is either empty or starts with a comment marker is ignored. Lines with curly braces open and close scope blocks, where scope blocks can optionally be negated whenever there exists a complement set, and can be nested. All other lines contain rules which together with the scopes applied under a certain context (e.g. user and groups database or data fragments) form the policies. Scope identifiers start with a ˜ sign to denote users and with @ to denote groups. Rules are not constrained by the language except for having to consist of 1 to n text tokens out of which the first one describes an abstract processing target with parameters 2 to n. Possible targets and contexts are defined by additional vocabulary, leading to a language flavour. In the next subsection, we will propose one suitable rule and context vocabulary.

Listing 1 defines the context-free grammar leading to the concrete textual syntax of FlexDDPL in Extended Backus-Naur Form (EBNF). For better readability, end-of-line tokens are not included.

Listing 1. FlexDDPL syntax in EBNF notation

```
policy        = { scoped-rule | comment } ;
scoped-rule   = scope-id " " "{" { scoped-rule
     | comment } "}" | rule ;
scope-id      = [ "!" ] scopecomp | scope;
scopecomp     = "@" | "~" scope;
scope         = unicode-name ;
rule          = module [ { module-option } ] ;
comment       = "" | "#" [ { ? unicode-char ?
    } ] ;
module        = unicode-name ;
module-option= unicode-name ;
unicode-name = ? unicode-alphabet-char ? { ?
    unicode-printable-char ? } ;
```

The sample in Listing 2 gives a first example of how flexible policies can be designed with the FlexDDPL syntax. On all data flowing through a cloud storage gateway, the `scaleimage` adaptation is applied. All members of the group developers are subject to a `obfuscatesource` adaptation as well, and additionally are constrained to storage providers from Europe. The storage controller is instructed that anybody's data except for the one of the head of sales will be stored twice at different providers.

Listing 2. Sample FlexDDPL policy

```
adapt scaleimage
@developers {
  adapt obfuscatesource
  store region europe
}
```

```
!~headofsales {
  control redundancy 200%
}
```

Listing 3 shows how data fragments can be introduced. They instruct the controller to split all incoming files into equally sized or weighted fragments.

Listing 3. FlexDDPL policy for data dispersion

```
fragment {
  control weight 75%
  store cloud
}
fragment {
  control weight 25%
  store periphery
  @developers {
    adapt compress
  }
}
```

B. FlexDDPL Vocabulary

There are two extensible sets of vocabularies to create powerful policies with FlexDDPL. The rule vocabulary determines which subsystems can be targeted by policies expressed in the language, and the context vocabulary determines which rules become active under certain contexts.

We propose the following subsystems for adaptive, dispersing storage gateways. All identifiers have been chosen on purpose to be usable both in noun and in verb form as the first expression token in rules.

- Store. Refers to the storage services as leaf nodes of the overall storage system. All services are assumed to be sufficiently described with non-functional properties including capacity, availability, region and price.
- Control. Refers to the cloud storage controller as branch nodes of the system. Along with splitting and encoding the data, the controller calculates and executes an optimal data distribution among the candidate services.
- Adapt. Refers to content adaptation operations both before entering the controller and within each node.

For the contexts, we propose the following identifiers:

- Users and groups. These are inherent to multi-user systems and, as previously described, denoted with the special prefixes ˜ and `@`.
- Fragments, declared by `fragment`. They allow a per-service differentiation between the parts of files stored on private and public resources. As cloud storage controllers offer nested fragmentation, this nicely coincides with nested contexts in FlexDDPL. Fragments can also be weighted to account for heterogeneity in the storage service capacities, and be given preferences to classes of storage services.
- Temporal and spatial context. When accessing the data at certain times or from certain locations, different rules may apply. For instance, the microformat expression `T:22:00-06:00` may give a temporal context for applying higher redundancy due to lower network transmission costs in this period.
- Data contents and metadata context. This is useful to distinguish public from security-sensitive files when they are tagged as such. Due to possible splits into fragments, this context is inherited into all subsequent nodes. An example would be `mime:message/rfc822` for e-mails.

We do not claim completeness over the proposed vocabulary. Indeed, new applications may arise and introduce new vocabulary, for instance privacy-justified dismissals.

Policies and policy engines might diverge over time because of the decoupling of syntax and vocabulary. As a recommendation, unknown vocabulary should be treated as a specification fault because reworking the appropriate parts or upgrading the engine is comparatively less work than the analysis of unexpected consequences of skipped or mis-interpreting rules and scopes.

IV. Data Distribution Gateway Architecture

Matching the flexibility of a data distribution policy language in a running system requires a coequally flexible storage gateway architecture. In our design, the gateway encapsulates both the policy creation and the policy evaluation and actuation components around the storage controller. Technically, it is therefore a policy decision point (PDP) and a policy enforcement point (PEP) at the same time. Before describing the integration of the policy components and their interaction with the gateway, we will first outline a basic gateway architecture and proposals how to implement it.

A. Basic Architecture

The core architecture without the policy-aware additions is centered around the data flow from applications to cloud storage providers and back. All data enters through a network service interface with a protocol suitable for integrating local network clients. Typically, this would be a WebDAV, CIFS/SMB or NFS proxy which is linked to an authentication and authorisation system. After passing the proxy, a storage controller handles the preprocessing, distribution, retrieval and recovery of data.

Administrators of the gateway require a channel to maintain the policy and other configuration. We assume an administration interface without restricting it to a certain technology such as web interface, configuration files or web service API to which remote clients can attach. This interface would typically be integrated with existing information services, in particular the storage service registry and the user and groups database as for instance LDAP/X.500 directory service or complex identity management solutions (IDM) with associated public-key infrastructure (PKI). The basic storage gateway structure is shown in Fig. 2.

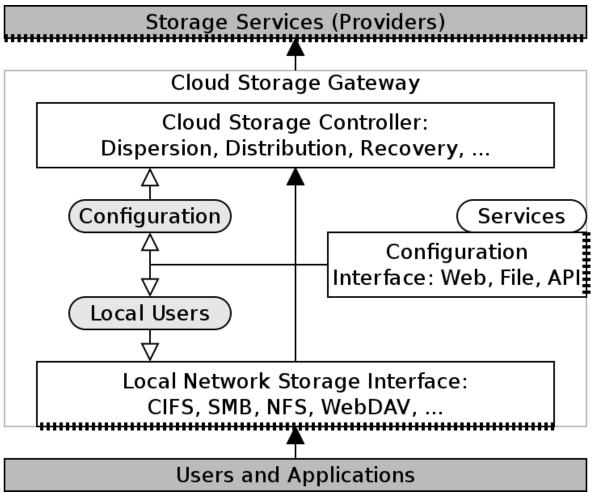

Figure 2. Architecture of a basic storage gateway for data distribution into well-described storage services

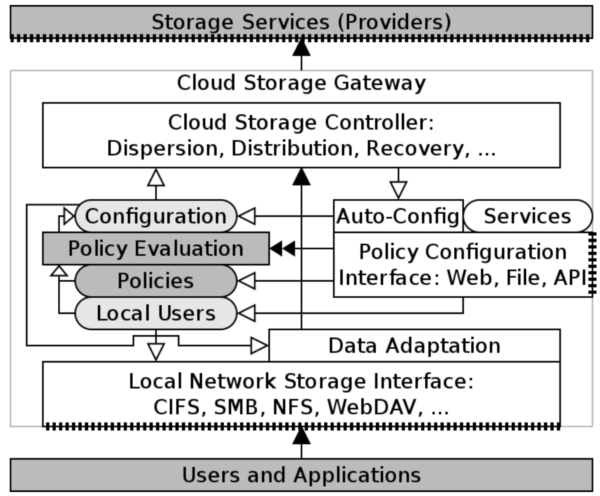

Figure 3. Architecture of an adaptive, policy-driven cloud storage gateway with autonomic reconfiguration

B. Policy-Aware Architecture Extensions

Along the data distribution path, an adaptation component is proposed due to promising results in existing gateways [12]. The controller and adaptation component can be separate pieces of software or interwoven in a pipeline architecture. The policy determines to which extent data will be adapted.

Furthermore, an autonomic configuration component assists the administrator in suggesting or even activating new providers and configurations. This component, too, operates under the guidelines and restrictions imposed by the policy.

The policy evaluation component parses the FlexDDPL document authored by the administrators and applies it under a certain context, which is primarily the users and groups database in the gateway. The application is indirect by influencing the configuration and activation of other components within the gateway.

Fig. 3 shows the compound architecture of the entire policy-aware, autonomic and adaptive data distribution gateway attached to applications in the front end and cloud storage providers in the back end.

C. Implementation

For the storage controller, we employ NubiSave, an existing optimal cloud storage controller [4], [13]. It represents a user-space filesystem (realised with the Linux FUSE kernel module) which disperses its data into directories. In most cases, these are mount points of other network filesystems. NubiSave primarily targets single-user single-device and, through configuration sessions which saved online, multi-device scenarios. Its locking support also allows for single-configuration multi-user environments by running a network service on top of the file system. Multiple configurations can be supported by mounting the controller's filesystem multiple times. This technique is used to realise the storage gateway.

For the adaptation of files and datasets coming across the gateway, we employ AdaptiveSyncer, an existing adaptive synchronisation tool [14]. It acts as an intermediate between two directories whose contents are synchronised with applied uni- or bidirectional adaptation. The storage gateway writes all incoming files to a temporary directory first, AdaptiveSyncer is triggered by the filesystem operation and produces adapted copies of the files, and finally removes the temporary files again. This intermediate step currently acts on whole files and hence undermines NubiSave's support for streaming large files through systems with small disks. An engineering compromise is hence to disable adaptation on large datasets and files.

Since no policy evaluation component could be found as existing software, we have implemented it as a Python library. It evaluates users, groups and fragment contexts. Advanced context including location, time and data contents is not considered in our implementation but can be extended following the given vocabulary structure. All `control` rules are transformed into appropriate statements in the storage controller's configuration files, whereas all `store` rules lead to a one-time invocation of the Online Service Signup Tool [15]. This tool needs two inputs: a requirements file and an identity to perform the service with. It is parametrised with a generated storage service goal description derived from the policy, and the identity assembled from the user database. All `adapt` rules are applied to an intermediate file system on which the adaptation tool works.

In our implementation of the administration interface, we have created a light-weight web interface with a custom user management sub-component. In each group, users who carry the administrator role can derive their own policies from the ones defined by groups in which the group itself has membership. This allows for both connected (i.e. hierarchical departments) and disconnected (i.a. hosted multi-tenant) installations.

V. Evaluation and Discussion

The evaluation of our contributions consists of two parts: First, we perform a language analysis of FlexDDPL according to standard guidelines on language construction. Second, we demonstrate the practical utility of the proposed implementation of a storage gateway through empirical methods. Both results will then be briefly discussed and summarised with an outlook on remaining challenges.

A. Evaluation of FlexDDPL

The evaluation of languages in general, and policy languages in particular, happens to follow either theoretic (semiotic) methods or practical (survey) ones. Both kinds of methods have been successfully applied to the recently presented Unified Service Description Language (USDL) [16], for example. For policies, guidelines on how to evaluate a Domain-Specific Language (DSL) exist. [17] The criteria presented therein are grouped into four categories: Language purpose, realisation, content and syntax, both abstract and concrete.

FlexDDPL fulfils most of the relevant criteria. It has been designed for a specific use case, has a textual representation, is kept simple through limited syntax elements and yet extensible in its rules and context vocabulary, and it allows comments. On the other hand, the language is not modular, doesn't offer more than one representation (serialisation) and also doesn't provide syntactic sugar.

The full set of 26 guidelines and how FlexDDPL follows them is shown in Table I. Not covered by this comparison is the tool support for authors and integrators. We have created a syntax highlighting rules file for the editor `vim` which can be obtained on the editor's website.

Guideline	Match	Guideline	Match
Purpose and Uses	v	Domain notation	o
Review	v	Description	v
Consistency	v	Distinguish	v
Representation	o	Syntactic sugar	-
Composition	-	Comments	v
Language reuse	o	Organisation	-
Types reuse	-	Balance	v
Minimalism	v	Style	v
Simplicity	v	Conventions	o
Specialisation	-	Syntax	-
Vocabulary	v	Layout	v
Non-Redundancy	v	Modularity	v
Efficiency	o	Interfaces	-

Table I

Domain-specific language design criteria (cf. [17]) matches by FlexDDPL: full match (v), partial match (o), no match (-)

B. Evaluation of a Policy-Aware Storage Gateway

For the empirical evaluation of a cloud storage gateway prototype, we choose the Debian GNU/Linux derivative operating system SPACEflight, a bootable demonstration system for service and cloud computing research [18], as a testbed. It already contains storage resource descriptions and user management applications, and its copy-into-memory-on-write design makes it suitable for deterministically repeatable experiments. On it, we install the NubiSave cloud storage controller [13] using the provided package, which automatically pulls in all dependencies. Along with it, we install the tools for adaptive synchronisation, for automated sign-up and for the policy evaluation, all three as interpreted scripts from their respective source code repositories. The experiment is performed on an Intel i5 quad-core notebook machine with x86_64 architecture, 4*3.6 GHz CPU frequency, 4 GB of memory and no swap space.

For each user and group found in the user database, one instance of the storage controller is mounted, which appears as a FUSE module mount point in the system. Each `store` rule is transformed into a service property requirement, eventually leading to attached storage filesystems. In order to find out about the scalability of this approach, we measured the filesystem initialisation times and the memory consumption of consecutive mount operations without attached storage filesystems. All measurements were performed with the provided shell-external `time` and `free` commands. After about 75 invocations, the operating system had to reclaim cached memory, and after 110 invocations, the memory resources were exhausted and the startup times became noticeable slow, as can be seen in the diagram in Fig. 4. We attribute most of this overhead to the FUSE module's Java code structure and expect this to be a major source for future optimisation. For smaller or medium enterprises, the current prototype would nevertheless already function as expected.

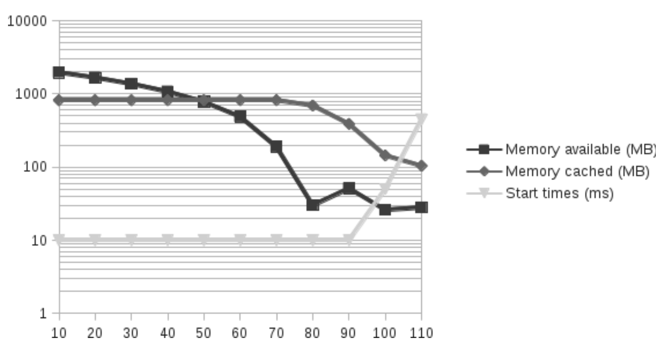

Figure 4. Resource overhead by using per-user/per-group user-space file systems in the gateway

The invocation of the evaluation engine itself is not noticeably performance-restricted. Over 1000 consecutive iterations on the sample from Listing 3, each iteration takes

30 milliseconds. This time does not yet include potential first-time efforts for the selection of and the sign-up to new services, which is heavily network-dependent, but typically in the order of a few seconds per account.

C. Discussion and Outlook

Cloud storage integration into multi-user environments benefits greatly from rigid, portable policies. With the Flexible Data Distribution Policy Language (FlexDDPL), such policies can be expressed independently from the storage controller and gateway implementation, and be mapped onto those as part of the administration processes. Our proposed storage gateway leverages this potential in an enterprise environment. We make our gateway implementation publicly available at [13].

The gateway subsystems have some potential for improvement. For example, recent works on long-term archiving in cloud storage environments differentiate temporally between high and low redundancy [19]. For security-sensitive environments, however, full replication is not a viable strategy to achieve high redundancy. Hence, we expect that time-driven erasure re-coders be part of storage gateways in the future. Furthermore, we expect more engineering work towards less resource-consuming implementations, and more research on appropriate service property vocabulary to be used in corresponding `store` rules. The design of FlexDDPL is future-proof with regards to the cloud storage evolution.

Acknowledgements

This work has received funding under project number 080949277 by means of the European Regional Development Fund (ERDF), the European Social Fund (ESF) and the German Free State of Saxony.

References

[1] S. Pearson, "Taking account of Privacy when Designing Cloud Computing Services," in *ICSE Workshop on Software Engineering Challenges of Cloud Computing*, May 2009, pp. 44–52, Vancouver, Canada.

[2] M. Borgmann, T. Hahn, M. Herfert, T. Kunz, M. Richter, U. Viebeg, and S. Vowé, "On the Security of Cloud Storage Services," Fraunhofer Institute for Secure Information Technology - SIT, Tech. Rep. SIT-TR-001, March 2012.

[3] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds," in *Proceedings of EuroSys*, April 2011, pp. 31–46, Salzburg, Austria.

[4] J. Spillner, J. Müller, and A. Schill, "Creating Optimal Cloud Storage Systems," *Future Generation Computer Systems*, 2012, accepted for publication.

[5] R. Seiger, S. Groß, and A. Schill, "SecCSIE: A Secure Cloud Storage Integrator for Enterprises," in *13th IEEE Conference on Commerce and Enterprise Computing, Workshop on Clouds for Enterprises*, September 2011, pp. 252–255, Luxembourg, Luxembourg.

[6] L. Pamies-Juarez, P. García-López, M. Sánchez-Artigas, and B. Herrera, "Towards the Design of Optimal Data Redundancy Schemes for Heterogeneous Cloud Storage Infrastructures," *Computer Networks*, vol. 55, no. 5, pp. 1100–1113, April 2011.

[7] O. C. Leonard, J. Nieh, E. Zadok, J. Osborn, A. Shater, and C. Wright, "The Design and Implementation of Elastic Quotas: A System for Flexible File System Management," Columbia University, Tech. Rep. CUCS-014-02, June 2002.

[8] M. D. Stefano, *Distributed Data Management for Grid Computing*. John Wiley & Sons, 2005.

[9] M. A. Amer, A. Chervenak, and W. Chen, "Improving Scientific Workflow Performance using Policy Based Data Placement," in 13^{th} *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, July 2012, Chapel Hill, North Carolina, USA.

[10] A. H. Anderson, "An introduction to the Web Services Policy Language (WSPL)," in 5^{th} *IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2004, pp. 189–192, Yorktown Heights, New York, USA.

[11] M. Ion, G. Russello, and B. Crispo, "Enforcing Multi-user Access Policies to Encrypted Cloud Databases," in 12^{th} *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, June 2011, pp. 175–177, Pisa, Italy.

[12] B. Nicolae, "On the Benefits of Transparent Compression for Cost-Effective Cloud Data Storage," in *Transaction on Large-Scale Data- and Knowledge-Centered Systems III*, ser. Lecture Notes in Computer Science, 2011, vol. 6790, pp. 167–184.

[13] J. Müller, J. Spillner *et al.*, "NubiSave Optimal Cloud Storage Controller," Software project, online: nubisave.org, 2012.

[14] A. Caceres *et al.*, "Adaptive Synchronisation Tool," Software project, online: serviceplatform.org/cgi-bin/gitweb.cgi?p=adaptivesyncer, 2012.

[15] C. Master, "Online Service Signup Tool," Software project, online: github.com/cloudmaster/osst, 2012.

[16] M. Matzner and J. Becker, *Requirements for a Service Description Language — Findings from a Delphi Study*. Springer-Verlag, March 2012, ch. 20, in: Handbook of Service Description - USDL and its Methods.

[17] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel, "Design Guidelines for Domain Specific Languages," in *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM)*, October 2009, Orlando, Florida, USA.

[18] J. Spillner, "SPACEflight - A Versatile Live Demonstrator and Teaching System for Advanced Service-Oriented Technologies," in *21st International Crimean Conference on Microwave and Telecommunication Technology (CriMiCo/*КрыМиКо*)*, September 2011, pp. 455–456, Sevastopol, Ukraine. [Online]. Available: http://serviceplatform.org/

[19] L. Pamies-Juarez, A. Datta, and F. Oggier, "RapidRAID: Pipelined Erasure Codes for Fast Data Archival in Distributed Storage Systems," arXiv:1207.6744v2, August 2012.