CLOUD
ACCOUNTABILITY
PROJECT

D:D-2.1 Architecture guidelines and principles

Deliverable Number:	D42.1
Work Package:	WP 42
Version:	Final
Deliverable Lead Organisation:	ATC
Dissemination Level:	PU
Contractual Date of Delivery (release):	M9 [changed to M15]
Date of Delivery:	19th December, 2013

Editor

Vasilis Tountopoulos (ATC)

Contributors

Stratos Tzoannos (ATC), Richard Brown (ATC), Thanasis Dalianis (ATC), Jesus Luna (CSA), Konstantinos Mantzoukas (CSA), Mohamed Sellami (Armines), Mario Sudholt (Armines), Jakub Sendor (SAP), Theofrastos Koulouris (HP)

Reviewer(s)

David Núñez (UMA), Bikash Agrawal (UiS)

A4Cloud

www.a4cloud.eu

Accountability For Cloud and Other Future Internet Services

FP7-ICT-2011-8-317550-A4CLOUD

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

The software development process can be approached from different angles and will potential consist of various steps, which vary depending on the objective and the nature of the final outcome. In secure software engineering, modern techniques and trends embrace the development of reference architectures as an essential roadmap for delivering scalable, re-usable and to-the-point solutions. The reference architecture should include all those architectural patterns and software artifacts that can be effectively instantiated in the context of particular business needs and provide the technology baseline for hosting specialised development.

Towards this direction, this deliverable offers a reference for reporting on the guidelines and principles to support the development of the A4Cloud Reference Architecture. It elaborates on the software development steps and how the Reference Architecture building is fed from specific stakeholder and technology-oriented requirements and drives the implementation of a multi-disciplinary environment to be actually used in various business domains. Based on the envisaged scope for inspiring all potential stakeholders in the (cloud) service provisioning chain on the way that accountability is instantiated across different contexts in the cloud and Future Internet applications, the document elaborates on the fundamental principles on how technology driven best practices can be shared among developers and analyses the guidelines for decision making on the technologies to build the cloud eco-system, with special focus on the accountability-based solutions.

In more details, the deliverable adopts existing standardised processes for managing the software development lifecycle, as a basis to place the work in WP:D-2 in the general picture of the A4Cloud project and describe the dependencies with the work performed in the other work packages. It, thus, defines the relationship of the Reference Architecture development, as part of the software design process, with the requirements elicitation and software implementation and integration phases. Subsequently, the report introduces the requirements for building the A4Cloud Reference Architecture by elaborating on the architecture principles and making an introduction to software development plan-based versus agile-based methodologies.

Although not digging into the details of the A4Cloud Reference Architecture, this document provides the roadmap from user-specific needs to software products, through a step-by-step realisation of the secure software development practices. In that respect, it acts as a reference manual for mainly the architecture designers, but also for the tool developers, on the scope of the A4Cloud Reference Architecture and its envisaged software development methodology, in accordance with the best practices and standards in the different scientific areas that the A4Cloud project is conducting research.

More specifically, the document presents the guidelines for the design of the Reference Architecture by analysing how it must be built both from a high level perspective and in detail. It, thus, identifies the needs for modelling those aspects of A4Cloud, which can contextually describe the reference architecture, and makes an introduction on the way that the work in the project may move from the conceptualisation view of the A4Cloud architecture to the actual implementation view, through presenting different perspectives (viewpoints), expected for the Reference Architecture.

An important aspect of the architecture guidelines evolves through the best practices found in the different standardisation efforts around the technologies that will support the proper instantiation of the Reference Architecture to the specific needs of the adopted business use cases. These practices aim to build consensus among project partners on the appropriate technology standards to be used across the project and ensure the interoperability of the developed technologies. In that sense, it complements the work in Task T:D-2.2 by offering a bottom-up approach to the analysis of architecture requirements and the development of the Reference Architecture.

To conclude, the report delivers the general design principles for delivering a meaningful and easy to understand and use cloud-based Reference Architecture, which will be elaborated in the future to address the needs for supporting accountability-oriented practices in the cloud service provisioning landscape. Furthermore, it presents the guidelines (including the appropriate software development approach and quality assurance procedures) for the development of the A4Cloud tools and their integrated view, as part of a business development scenario.

Table of Contents

1 Definition of problem

The role of WP:D-2 is to deliver a sustainable Reference Architecture for accountability-based approaches in protecting data in the cloud and future Internet services. The architecture must identify the appropriate conceptual, logical and technical pillars that enable potential stakeholders in realising this architecture and instantiating it when implementing specific business cases. Thus, this section introduces the scope of the A4Cloud Reference Architecture and how the activities for specifying it fit to the standard processes for delivering commercially strength software releases.

1.1 Introduction

The specification of the general principles for system and software development and the guidelines for transforming user-specific needs to software products and/or systems have been extensively discussed in the ICT community for years. The exact process to be followed varies, depending on the nature of the software and/or system, as well as the level of involvement from the stakeholders' side.

This WP:D-2 deals with deriving a Reference Architecture for accountability-based approaches in offering services and application in cloud ecosystems. This Reference Architecture should be the result of a standardised approach in software engineering domain and should facilitate architectural design instantiation to a wide range of application domains towards effective information stewardship and data protection in the cloud and other Future Internet environments. Thus, the principles and guidelines for delivering the A4Cloud Reference Architecture must be placed in the context of standardised processes on software and system lifecycle.

Numerous references in the bibliography introduce the process activities for delivering software products and systems. The ISO/IEC 12207:2008 [1] is the IEEE standard for the definition of the software lifecycle processes, which should be followed in the design and implementation of software. These processes are realized in the context of the Software Implementation Process of the ISO/IEC 12207:2008 and are summarized in the following:

- Software Requirements Analysis Process: this process is used to elicit requirements for the software implementation. The relevant activities for the accomplishment of this process are served from the work conducted in WP:B-2, WP:B-3, WP:B-4, WP:B-5, WP:C-2 and WP:C-3, which materialise the requirements collected from different stakeholders in the various technical, legal and socio-economic disciplines that the project aims, including the business areas that the use cases lay on.
- Software Architectural Design Process: this is the process for designing the software, which implements the requirements that will be further assessed against them after development has been delivered. The scope of this document lays on the parts of this process.
- Software Detailed Design Process: this process is based on the previous one and extends it to deliver the details of the architecture design. This process will be elaborated in the other deliverables of WP:D-2.
- Software Construction Process: this process implements the design of the previous process. The guidelines for accomplishing the work in this process are drawn in this document, while the actual development work is performed in the respective WPs of Streams C and D.
- Software Integration Process: the main target of this process is to deliver a unified view of the software components and units being implemented in the previous process. The guidelines for accomplishing the work in this process are drawn in this document, while the actual integration effort is part of WP:D-7.
- Software Qualification Testing Process: this process exploits the integrated output of the previous process and performs compliance check of the implemented software against the defined requirements. This is actually part of the work to be conducted in WP:D-6 and WP:D-7.

A similar approach is presented in [2], in which the software engineering is realised through four fundamental activities, namely software specification, software design and implementation, software validation and software evolution: The software must evolve to meet changing customer needs. In terms of the scope of this document, the approach described in [2] does not introduce any significant differences. Thus, this report partially address the first two activities for software specification and design and aims to deliver the general design principles for delivering a meaningful and easy to understand and use Cloud Accountability Reference Architecture, as well as the guidelines (including the appropriate software development approach and quality assurance procedures) for the development of the A4Cloud components and their integrated view, in the context of business use cases, as they are introduced in WP:B-3.

The final goal of this document is to offer a baseline reference for the architecture designers and the tool developers on the scope of the A4Cloud Reference Architecture and its development methodology, in accordance to the best practices and the standards in the research areas of the project. It will, also, enable building consensus on the technology standards to be used across in the project and ensure the interoperability of the developed technologies. To this end, this document complements the work in Task T:D-2.2 by offering a bottom-up approach to the analysis of architecture requirements and the development of the Reference Architecture.

1.2 The scope of the Reference Architecture

The definition, adoption and effective use of a Reference Architecture is an integral part of the software lifecycle processes, which introduced in the previous section, as it provides the design specifications, which translate user level requirements and needs to deployable software components. The Reference Architecture has been introduced in various sources [3][4][5][6][7][8][9][10][11] to define the scope for instantiating it in different business contexts and the objectives in many IT domain-specific problems and application fields.

In A4Cloud, we adopt the following definition for the Reference Architecture: "*A Reference Architecture is, in essence, a predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed and proven for use in particular business and technical contexts, together with supporting artifacts to enable their use*". As such the goal of the A4Cloud Reference Architecture is to provide an abstract and powerful model for designing accountability in modern cloud and future Internet ecosystems. It, actually, invades as an essential step towards addressing the requirements of the target stakeholders by defining the architecture vision and capabilities and delivering a roadmap to implement such requirements in specific cases, aligned with certain business goals.

Many such studies have revealed the necessity for building reference architectures for a variety of reasons. All these reasons can be efficiently summarised in the work performed by Architecture Forum Meeting, as reported in [8]. As it can be seen there, the pillars for building reference architectures are concluded in the business context of use (*increasing complexity, scope and size of the system of interest, its context and the organizations creating the system*) and the effective handling of dynamic changes in ICT environments (*increasing dynamics and integration: shorter time to market, more interoperability, rapid changes and adaptations in the field*), which may have particular impact on the acceptance and maturity of the proposed software solutions.

The A4Cloud Reference Architecture aims to have a multiple effect on the cloud service and Future Internet (FI) markets. As cloud and distributed computing in FI is widely spread in a multidimensional business environment, organisations dealing with personal and corporate data must be accountable on the use of such information to deliver high quality results. In such a complex environment, the A4Cloud Reference Architecture targets to *achieve interoperability between various evolving systems* and leverage the development of secure and trusted information systems and applications, through offering a set of architectural guidelines, which facilitate the compliance to specific design principles.

The ultimate scope of the A4Cloud Reference Architecture is to offer all possible stakeholders in the (cloud) service provisioning chain guidance on how accountability can be implemented across different contexts in the cloud and FI applications and share technology driven best practices and tools for delivering accountability-based solutions to end users.

1.3 Structure of the Document

The structure of this document is as follows:

- Section 2 introduces the requirements for building the A4Cloud Reference Architecture by elaborating on the architecture principles and making an introduction to software development plan-based versus agile-based methodologies.

- Section 3 presents the guidelines for the design of the Reference Architecture. This section elaborates on the way that the A4Cloud high level and detailed design will be achieved. It, thus, analyses the needs for contextual and data modelling in A4Cloud and presents candidate languages for process modelling, as an introduction to move from the conceptualisation view of the A4Cloud architecture to the actual implementation view. This section concludes with the definition of the different viewpoints that should be expected for the Reference Architecture.
- Section 4 analyses the way for the instantiation of the Reference Architecture during the main development phase of the A4Cloud project. It, more specifically, refers to the different areas of work during the implementation of the architecture and the standards that are pre-selected to support the development activities. Thus, it presents candidate standards and technologies that relate to the A4Cloud work, in the areas of Policy and Identity Management, Software and User Interface Development, Application Server and Persistency technologies. Moreover, it focuses on interoperability standards and technologies and makes an introduction to the approach for loosely-coupled integration to be performed as part of the use case instantiation in WP:D-7.
- Section 5 presents the Software Qualification Testing Process and focuses on the ISO/IEC 25010:2011 standard, as the most widespread reference model for technical assessment and quality assurance.
- Section 6 concludes this document.

2 Requirements on the design of the Reference Architecture

The design of the Reference Architecture is based on requirements arising from the target stakeholders that are going to instantiate and use the proposed framework (functional level requirements), as well as key architectural guidelines and design principles (non-functional level requirements), which should be respected in order to ensure the sustainability of the Reference Architecture and its wider acceptance by market sectorial developers and system engineers.

The design of a Reference Architecture is driven by principles, which try to answer questions like the recommendations raised in [11]. A Reference Architecture should have a clear target audience or might be presented from different views [12][13] in order to be consumed by different stakeholder groups. In most cases, these groups should show an expertise in certain sectorial business and the Reference Architecture can, subsequently, address the requirements of specific application and business domains. As per [11], it is questionable that Reference Architectures can expose those quantitative data/evidence, which would potentially be useful as metrics for the evaluation of the Reference Architecture and the assertion on whether it can be appropriate and have direct added value if applied to certain business domain.

Having the above considerations in mind, the work for the design of the A4Cloud Reference Architecture in this WP:D-2 is based on the outcome of the project Software Requirements Analysis Process. This is the process for eliciting the A4Cloud requirements, which is covered by the work performed in WP:B-2, with involvement of other WPs, being allocated to Streams C and D. The resulting accountability requirements are to be classified and associated with the accountability practices that adopt the relevant mechanisms to implement the Accountability Model in WP:C-2. It must be noted that the detailed specification of the Reference Architecture is by principle subject to the adoption of certain software development methodologies.

The relation of the A4Cloud Reference Architecture development to the other A4Cloud WPs is presented in Figure 1. Specifically, the A4Cloud Reference Architecture should be designed in such a way that the accountability practices defined in WP:C-2 can be associated to specific software components implementing the logical functionalities. The latter materialise the accountability practices and embrace the accountability mechanisms. The practices should elaborate on the expected behaviour of the accountable organisations, as well as the interactions being held in a complex cloud service chain ecosystem. From an operational point of view and in the general case of a cloud service delivery example, accountability means that the involved cloud actors should be able to meet the following goals:

- Define data stewardship in the cloud service chain, so that responsibility for the treatment of personal data and confidential data is assigned. At this point the particular mutual agreement over the cloud actors on the exact data governance processes should be established.
- Ensure implementation of the appropriate actions, by i.e. developing a continuous reporting and monitoring framework that enables accountability holders to make assessment on whether the agreements are being respected.
- Demonstrate compliance over regulations, policies and practices.
- Implement a remediation framework that delivers the corrective mechanisms, which are necessary in case of failure to act properly and in accordance to the regulations and the agreed policies.

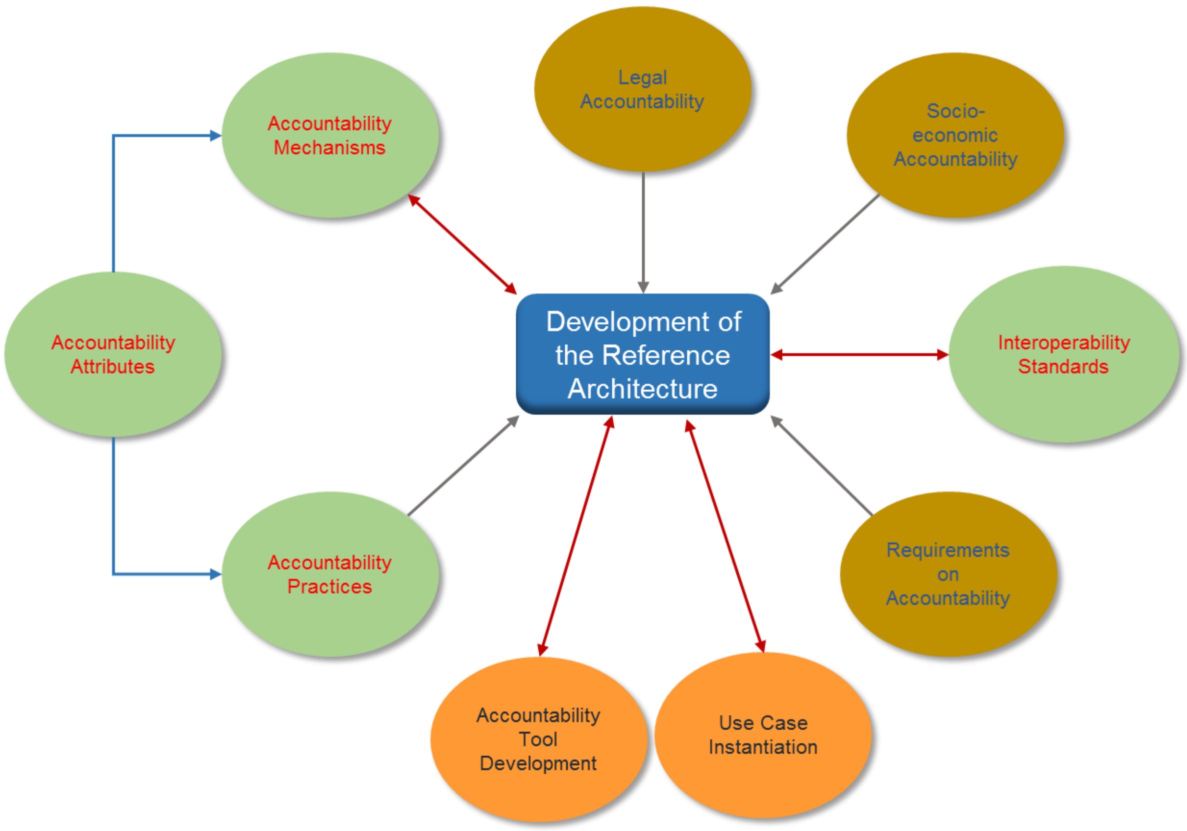
Accountability Mechanisms
Legal Accountability
Socio-economic Accountability
Accountability Attributes
Development of the Reference Architecture
Interoperability Standards
Accountability Practices
Requirements on Accountability
Accountability Tool Development
Use Case Instantiation

Figure 1: Relation of WP:D-2 work to other WPs

2.1 A4Cloud Design Principles

The requirements for the software development process are enriched with general design principles, which affect the way that the Reference Architecture should be built. These principles stem from the objectives of the architecture and the considerations presented above. On top of that, A4Cloud targets cloud and distributed Future Internet environments, and develops on top of existing work different Standardisation Developing Organisations (SDOs), (see [14][15][9][16][10]). This means that fundamental architectural design principles that have been followed in the development of the Reference Architectures of these SDOs should be respected in the A4Cloud roadmap as well. Thus, the development of the A4Cloud Reference Architecture must be guided from the following principles:

Architecture Principle 1: Simplicity on the Reference Architecture

The Reference Architecture targets wide audiences in the cloud service supply chain. This means that the depiction of the architecture should be comprehensive to different stakeholders, with as much information as possible provided on the architecture functional blocks. The use of different views for the Reference Architecture is appropriate, so that commercial end users can understand the architecture concept and the logical functions to be implemented, business IT users can realise the behavioural and procedural aspects of the accountability practices, technical development teams can decompose the logical view into software classes that facilitate the functionality scope and operational and service execution teams can elaborate on the deployment analysis of the necessary software and hardware resources. Simplicity over the definition and depiction of the Reference Architecture must be ensured to enable these target stakeholders easily instantiate it towards addressing their needs.

Architecture Principle 2: Robustness through multi-layered approach

While simplicity means that the Reference Architecture must be seen from different stakeholders' perspectives, robustness relates to the provision of functions and software modules with a certain level of independence. This can be enabled through following a multi-layered approach, which results in decoupling on roles' responsibilities and minimising the bottlenecks in the architecture customisation to specific business solutions. The distinction to layers offers independency in the execution of software components through loosely coupled interfaces, which smoothens the maintenance process in order to effectively handle bug fixing and extensibility to new functionalities on demand.

Architecture Principle 3: Addressing Accountability Practices

A4Cloud Reference Architecture should implement the concepts of the Accountability Conceptual Framework. As such, the WP:C-2 accountability attributes and mechanisms should drive the development of the architecture and complement the requirements arising from the Software Requirements Analysis Process. It, thus, takes into account the compliance with data privacy standards and regulations and incorporates privacy preserving, security and trust functions for policy-based exchanging of information between the cloud actors.

Architecture Principle 4: Conformance to existing standardisation efforts

A4Cloud is built upon existing efforts by different SDOs. The examination of current standards and the extension of them where this is necessary should be a key objective for the A4Cloud Reference Architecture. In that respect, the work performed in WP:A-5 is of high relevance for the specification of the A4Cloud Reference Architecture, since existing standards will feed the work here and the analysis of the architectural blocks in A4Cloud will reveal any missing gaps in standardisation to be fed back to WP:A-5. The environment that A4Cloud must operate has already established a certain level of maturity, with the cloud market to evolve on the distributed system engineering paradigm (like Service Oriented Architecture and Model View Controller architecture patterns) and deliver solutions for the cloud service and resources management, monitoring and maintenance that can be leveraged with accountability models. Thus, all the provisions of the architecture, such as machine interfaces and data information formats, must conform to relevant industrial standardisation activities, be compatible to technological advances and exploit best practices in the accountability and privacy protection domain, in order to maximise the interoperability potentials.

Architecture Principle 5: Technology Neutral

A4Cloud Reference Architecture must be technology agnostic, meaning that the proposed logical and functional views should be transparent to the underlying technologies and tools that are finally selected to implement a specific instance of the Reference Architecture. Thus, when defining the different views of the Reference Architecture and proposing best practices for accountability adoption, there should be no inference to particular technologies to be used, maintaining in that way flexibility over the development of accountable architecture instances.

Architecture Principle 6: Security-by-Design Architecture Operations

When moving away from traditional architecture approaches, the cloud and distributed computing examples raise questions on whether the proposed architectural designs can ensure the necessary levels for security and trust. The delegation of functionalities to third party vendors introduces a controversial bottleneck in achieving business continuity. Thus, A4Cloud Reference Architecture have to identify independent functional blocks that provide verification and validation checks towards disclosing security leakage and strengthen the business to IT alignment since at design time level. This can potential be achieved through performing risks and threats analysis as the Reference Architecture is built and conducting vulnerability testing over the derived instances of the architecture to the target business domains.

Architecture Principle 7: Scalability in the development of sustainable functionalities

Scalability is considered as the number of concurrent users, who can simultaneously access the solution services and the data volume, which can be handled in the software processes.

Architecture Principle 8: Support economies of scale

The Reference Architecture must be designed to support scales of economies by maximising the reusability potentials for the functional building blocks and their associated software products. Common processes, flexible data structures and generic functions must be grouped into patterns that implement abstract models of these processes and offer opportunities to maximise the efficiency of accountability-based cloud operations by minimising the development costs and reducing the time needed for new business to go to the market.

Architecture Principle 9: Reside in the secure software development and cloud service lifecycles

A4Cloud Reference Architecture must support the accountability-based design, threat analysis testing and secure execution of software for cloud and other Future Internet services. Thus, it should hold in a parallel pane and propose functions residing across the generic proposed states of the typical cloud service lifecycle, as it is proposed in many SDOs, such as in [17][18][19], and the stages of secure software development, like the one proposed in [20][21]. As existing cloud-based software and services solutions adopt such kind of lifecycle states, the conformance of the A4Cloud Reference Architecture to the secure development and cloud service lifecycles embraces the availability of deployment of this architecture to multiple concrete solutions.

2.2 Software Development Methodologies

The delivery of the A4Cloud Reference Architecture is tightly coupled to the adopted development methodology. The level of detail that is required in the definition and development of the architecture depends on whether the resulting software components follow one or another software development methodology family, so the approach that is required to meet the architecture principles might differ a lot [22]. In that respect, in this section, we introduce different development methodological approaches and conclude with the one that A4Cloud can follow in order to meet the set architecture principles and implement the guidelines suggested in the next sections.

A software development methodology is a collection of software development principles and practices, which enable development teams to engineer the software development processes, by introducing the principles to structure the software needs, plan the development of the specific software and take control over the software releases so that they can be tested and used by the envisaged stakeholders.

A number of available software design methodologies have to be examined [23], in order to come up with the methodology that better fits to the A4Cloud objectives. All of the proposed methodologies in the literature expose specific benefits and drawbacks and target to different level of detail the design and specification of the appropriate architecture to guide the software development.

The existing software development methodologies are mainly classified into two main families of methodologies:

- Plan-based software development methodologies; in this family, a number of methodologies are proposed, which focus on the software life cycle and the clear distinction between the phases constituting this life cycle.
- Agile software development methodologies; in this family of software development methodologies, requirements elicitation and software quality assurance are fundamental parts of the software design and implementation and these phases run in parallel to produce a high quality output to the target users.

Plan-based methodologies

This family is based on the prediction of the software lifecycle and it counts on a strong requirements specification documentation, which actually covers the development process. Such methodologies do consider for iterations, but the success of the methodology is based on the well-defined outcome of one phase before proceeding to the next one. The documentation of the software is another fundamental characteristic of this group of software development methodologies.

A number of plan-based software development methodologies exist [24][25][26]. From them Waterfall Model and Rational Unified Process (RUP) are the most well-known ones.

Waterfall Model [27] is a sequential flow-based approach for software development, which defines the phases for analysis, specification, design, coding and testing. The steps to accomplish the software development can be pre-defined, based on well described documentation and the phase execution follows a strictly sequential flow, in which one step is initiated only when the previous one has been finalized and produced a clear output. Waterfall methodology counts on the proper and accurate definition of one phase, thus it does not involve any iteration. This means that any problems must be foreseen early in advance and subsequent changes and/or revisions on the requirements cannot be handled in a later phase. The Spiral Model is the evolution of the traditional Waterfall Model, as it does the iteration perspective between the different phases of the software development life cycle.

The main advantage of Waterfall Model is that fact that it is easy to be realized by all the involved parties in software development, from the developers and engineers to the end users. Planning and detailed milestones can be an asset when dealing with complex and large systems, while error detection and problem identification come at an early stage. On the other hand, the lack of iteration is a main drawback giving few or no flexibility in a continuously changing environment.

RUP [28] was first invented by IBM and it provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget. RUP is a methodology targeted to the management of object-oriented systems and it is the most widely spread methodology for information system development and implementation world-wide. RUP defines four phases, namely Inception phase (requirements gathering), Elaboration phase (System Design), Construction phase (System Implementation & Customization) and Transition phase.

In RUP, initial requirements registration is followed by detailed system architecture and the overall technological environment design (process particularly important in any project), succeeded by the initial design of the components and subsystems. The integration process of a systemic design comes accompanied by the completion of the architectural design which aims to validate the specified options and the first demonstration of the architecture. The evaluation of this prototype provides an initial feedback on the redesign of the next prototypes. The entire process leads to the initial functional prototype implementation and its evaluation by the users. This is especially important considering the fact that users will be able to "use" a system very close to the desired and provide additional requirements and valuable feedback on all system quality. Eventually a new cycle of design, implementation and integration is carried out, taking into account on the one hand the evaluation results of the first prototype and on the other hand the original design of the system.

The main advantage of RUP with respect to Waterfall Model and other plan-based methodologies is the fact that changing requirements are considered and integration is performed progressively. This means that the development process itself can be improved and refined, while iterations reveal the need for changes that should be addressed in the next iteration.

Other plan-based methodologies include the Rapid Application Prototyping (RAP) and the Joint Development Methodology (JAD).

Agile methodologies

Agile software development (ASD) [2][29] is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organising, cross-functional teams, addressing the development efforts performed in the various stage of a project (like the A4Cloud streams C and D).

There are many specific ASD methods. Most of them promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project. ASD goes beyond traditional software development processes (such as Waterfall) and exploits an evolutionary method that is an iterative and incremental approach to software development and integration. Thus, the requirements and design phases are iteratively met with the development phase to incrementally produce system software releases, which can be assessed over the suitability, the maturity and the immediate business value. On top of them, ASD foresees an intense testing phase, in which the unit testing is achieved from the developer’s perspective and the acceptance testing is conducted from the customer’s perspective. Thus, the major difference with respect to the iterative approaches of plan-based methodologies (like RUP and Spiral) is the fact that requirements and testing are part of the actual development iterative process and the target stakeholders can be progressively involved in the development process aiming to deliver high quality software.

"The best architectures, requirements, and designs emerge from self-organising teams" [30] which can be effectively grouped in an agile-based fashion. ASD manifests [31] that "*working software is the primary measure of progress*". Thus, agile methodology approaches the requirements of a software environment, like the cloud market, iteratively by frequently delivering working software prototypes. These prototypes enable the project development and business teams to work together and maximise the quality of the produced output.

ASD can be suitable for the development strategy of an ICT solution, mainly because:

- A parallel process between the development of the planned software solution and the verification of the requirements can be followed, leading to business oriented people to actively participate in the specification of use cases and the evaluation of the system developments and provide valuable feedback in an iterative way;
- The work on individual and independent development fields is split among small groups comprising the separate development teams;
- As a ready to the market solution is envisaged rapidly, the solution can benefit from frequent releases to align the work done among the individual teams;
- The producing releases can be exchanged among senior technical teams and business oriented groups to evaluate the effectiveness of the solution in real business situations.

Extreme Programming (XP) [29] and SCRUM [29] are the most popular agile-based software development methodologies. Such methodologies can decrease the time required to produce releases and engage the customer in the development process, maximising the possibilities for a high quality output. On the other hand, the effectiveness of these methodologies is tightly coupled to the proper communication between business end users and engineering teams, while the lack of specific documentation might put the final acceptance at risk.

Selection of Software Development Methodology

A4Cloud is an R&D project that aims to promote innovation, based on actual market needs. The project has a well-defined time plan that defines phases to be followed by the different project teams to deliver efficiently and timely results to target stakeholders. These phases include concrete requirements analysis and architectural specification that should drive the implementation of frequent software prototype releases to be assessed in the context of real use case instantiation paradigms.

It is apparent that a plan-based software development methodology should by principle be followed in A4Cloud to deliver the necessary software perspective of the accountability in the cloud and future Internet services. The identification of phases has the meaning of progressively delivering a concrete outcome per project phase and the phases are iteratively adopted to assess and refine the results of the preceding phase. However, A4Cloud is a living R&D effort that is impacted and impacts the environment, consisting of the cloud markets and stakeholders that will have to adopt the accountability concept and align their business processes to a more secure and reliable context towards protecting personal and corporate data in the cloud. As such, the different plan-based phases are enriched with agile-oriented practices to offer an incremental approach for rapid software development when moving from phase to phase and from iteration to iteration. The need to inject agile into plan-based methodology arises from the fact that A4Cloud envisions break-through solutions, thus the innovation dimension needs to be reflected in the actual development phase and goes beyond pre-defined and probably isolated architecture specifications.

In more details, the A4Cloud project as a 42 months research and development effort should deliver state-of-the-art solutions in accountability for the cloud. In order to minimise the risk from finally providing obsolete tools, the methodology that A4Cloud follows extends the typical requirements, development and evaluation cycle, which is illuminated with agile sprints for the development of breakthrough research and the instantiation of the research to business use cases in order to leverage the adoption of accountability-based practices in protecting personal data in the cloud. One should not disregard the fact that multiple disciplines are grouped together in A4Cloud, thus prompt interactions between the involved stakeholders will safeguard the quality of the final outcome.

3 Guidelines on the Design of the Reference Architecture

As stated in Section 2.1, the A4Cloud Reference Architecture is based on a set of architecture principles. These principles affect the way that the architecture will be designed and detailed towards specifying the different architectural characteristics of this Reference Architecture.

3.1 Advancing Standardisation Efforts

As part of the Software Architectural Design Process, the design of the Reference Architecture should follow the Architecture Principle 4:. An important guideline towards this direction is the adoption of existing standardisation efforts with respect to the different views of a cloud reference architecture with emphasis on the security, trust and privacy problems. As such, we have studied the work done in the existing reference architectures that the different Standardisation Developing Organisations (SDOs) have delivered so far in the research disciplines of the project. The set of A4Cloud requirements is the basis for placing the A4Cloud Architecture in the context of the existing reference architectures.

Existing reference architectures can be found in many SDOs. The accountability aspects are not adequately addressed in the current architectural standards, but the latter can constitute the starting point in order to derive a proper view for an accountability oriented Cloud Reference Architecture. Depending on the focus of each SDO, the reference architectures can cover general aspects of the cloud computing domain, while some others elaborate more on the security perspective on the cloud computing. In all cases, security and risk assessment are arisen as parallel vertical panes to the traditional purely technological layered reference architectures for delivering the functional view of the envisaged solution.

During this period, the following architectures have been considered:

- The security architecture of the International Telecommunications Union (ITU) [14]: this architecture proposes a three layered security architecture, which is complemented from three security planes and envisions to enforce security in eight dimensions.
- The Cloud Management Architecture of the Distributed Management Task Force (DMTF) [15]: this architecture sets the security context as integral part of cloud actors' interaction patterns.
- The National Institute of Standards and Technology (NIST) conceptual reference architecture [9]: this architecture defines an actors' based conceptual model, in which security is seen across all layers of it.
- Cloud Security Alliance (CSA) Enterprise Architecture [16]: this complex architecture moves one step further and brings together the business and technical operational views with the security and risk management considerations. Again security is seen as a complementary plane across all functional layers of the IT-oriented reference architecture

3.2 Reflecting Accountability needs

The A4Cloud Reference Architecture must showcase the accountability-based approach in the cloud service chain. The defined accountability practices embrace the context of use of the A4Cloud tools and drive the interactions among them and with external tools and processes to build meaningful business paradigms. In line with Architecture Principle 3:, the Reference Architecture should work around the following.

Contextual modelling

In this initial phase of the architectural design process, the A4Cloud work in this WP:D-2 should align the design with the accountability practices, as they are derived from WP:C-2. The practices complement existing and standardised general cloud computing use cases, as they are defined in the different SDOs and have been extensively presented in their relevant publications (e.g. see [17][18][19]). NIST in [32] defines 25 use cases classified into three groups: cloud management, cloud interoperability and cloud security. OMG adopts a more abstract way to introduce 7 use cases [18], targeting to highlight the interactions of the cloud users with the cloud (as an actor) and between the different clouds. DMTF in [18] has produced a list of 14 use cases in this direction. All these use cases are identified in the course of the main stages of a cloud service lifecycle, which means that the contextual modelling of the A4Cloud Reference Architecture satisfies Architecture Principle 9:.

Whether trying to map or link the accountability practices to some or all of these use cases, it is apparent that the final goal is to lead the work for the realisation of the context on which A4Cloud should operate. And this part includes both A4Cloud specific products (the Streams C and D tools) and external tools that need to be adopted in the project. In that respect, the accountability practices can be realised in the context of actual business processes met in everyday life tasks.

Data Structure modelling

Moving from the contextual modelling of the Reference Architecture, an important step towards reflecting the accountability needs in a cloud service eco-system is the data structure modelling. The A4Cloud Reference Architecture must work on the definition of A4Cloud specific information structures, which implement the accountability concept across the different interactions held in the accountability practices. Thus, these structural data forms are going to be implemented in the adoption phase of the Reference Architecture to accomplish the customisation of the accountability practices in a certain business context.

Data structure modelling puts together the piece of information that the different design functional blocks should utilise as input, produce as output and communicate as information objects mainly to other blocks. At this stage of the project, the work in D-2 will be able to produce an abstract view of the data models needed in a Reference Architecture, which will be then instantiated, based on the use cases and the work in WP:D-7.

From a provisional analysis of the work in the different streams, the following data structures can be identified so far (the list is only an example, and does not provide the exhaustive list of common data structures to be produced in A4Cloud):

- The risk representation model, which models the risks associated with the various cloud stakeholders and it will be defined in WP:C-6.
- The trust representation model, which models the trust relationships among the cloud stakeholders in a cloud service provisioning chain and it will be defined in WP:C-6.
- The socio-economic representation of the risk and trust models, detailing on the provisional value of risks and trust models in the accountability chain, as defined in WP:B-5.
- The cloud service specification model (generic non A4Cloud specific model), which describes the functional and security characteristics of a cloud service. The service specification is closely related to the underlying service execution infrastructure.
- The data policy model, which describes the provisions of the policy in a dedicated policy language, as it will be described in WP:C-4 and WP:D-3.
- The accountability metrics model, which is a representation of the measurements for calculating the values of the accountability attributes, as it is described in WP:C-5.
- The evidence representation model, which elaborates on the way that the collecting log data is analysed to offer the reasoning behind an event analysis, as it is described in WP:C-8.
- The violation notification model, which presents the way that the stakeholders are notified in case of policy violations detection, as it is described in WP:D-4.

3.3 Delivering a multi-layered approach, exhibiting simplicity and transparency

Simplicity must always guide the development of reference architectures. A4Cloud targets stakeholders of multidisciplinary expertise and business focus, thus Architecture Principle 1: is of high priority when developing the architectural design of A4Cloud. Different stakeholders must have the same level of realisation on what A4Cloud Architecture is delivering. Thus, the work on the production of the A4Cloud architecture design patterns must be performed aiming to complementary perspectives, such as:

- The organisational perspective, in which the Reference Architecture reflects how the design satisfies the organisational processes and policies that can be realised from the different business, business continuity and regulatory teams.

- The semantic perspective, in which the Reference Architecture identifies the common data representations that are to be communicated along all the design blocks of the architecture. This perspective is more appropriate for both IT business and technical teams.
- The technical perspective, which highlights the need for the adoption of common interoperable technological standards, so that the communication of the design blocks can be achieved. This perspective is necessary for technical and IT expert teams mainly.

By providing different views of the A4Cloud Reference Architecture, the guideline of simplicity can be achieved. These views not only facilitate the above mentioned perspectives, but also assist in deriving a set of architectural assets (i.e. functional blocks, architecture roles, etc.) that can be eventually mapped to software tools and respective information blocks. This analysis will be based on the mapping of the Reference Architecture assets to cloud stakeholders' roles and will, thus, showcase specification of the interoperability requirements for achieving accountability among these roles, as they have been identified in WP:C-3 work.

The different architectural perspectives are then in line with Architecture Principle 6: on driving the materialisation of the functional characteristics of the Reference Architecture to independent architectural blocks. By distinguishing on the necessary functions and delivering various views of them to be realised from different stakeholders, a certain degree of control on the necessary architectural processes can be maintained, since specific high and low level functional responsibilities are attributed to separated architectural roles, either being the users of the Reference Architecture or the individual blocks of the architecture itself.

Following Architecture Principle 2:, the majority of SDOs proposes a layered approach for the delivery of Reference Architectures. As such, A4Cloud will build the Software Detailed Design Process by analysing the Reference Architecture from different views, as shown in the following lines.

Logical View

This view will provide an abstraction of the A4Cloud envisaged outcome, showing the main cloud objects and their associated classes, detailing the functionalities arising from the A4Cloud requirements and the contextual modelling of the A4Cloud Reference Architecture. The logical view can use either Model-View-Controller (MVC) based or Service-Oriented-Architecture (SOA) based design patterns and can be realized either as a layered logical architectural design or a logical grouping of functionalities as done in the NIST RA document [9].

A rough split into four functional layers is advisable, as follows: user interaction and service/application presentation, interface access, business and service logic, network and resource management.

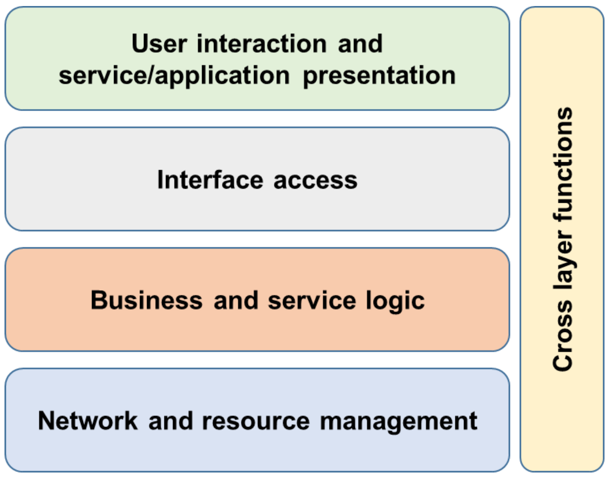
User interaction and service/application presentation
Interface access
Business and service logic
Network and resource management
Cross layer functions

Figure 2: An example of a layered logical view

The user interaction and service/application presentation layer mostly refers to human computer interaction design patterns. At this layer, the recommendations from WP:C-7 about the definition of principles and guidelines for human centred design of usable and privacy preserving User Interfaces are to be considered.

The access interface layer facilitates the interoperability requirements for accountability and hosts design patterns that facilitate to both functional and management service capabilities.

The business and service logic layer performs the self-contained tasks that facilitate the accountable service chain processes.

Finally, the network and resource management layer is more on the actual cloud infrastructure resources necessary to support storage and communication aspects of accountable service chain processes.

Process View

The logical analysis of the A4Cloud Reference Architecture will be placed into cloud-based processes for implementing accountability practices, as per Architecture Principle 9:. This view should demonstrate the interactions among the logical objects of the previous view.

Components Development View

Digging into the details of the A4Cloud work, this view will decompose the logical view into software components. The detailed API specification of each component should be defined here. Based on the software development methodology (see Section 2.2), the components will be subsequently implemented in the next Software Construction Process and be incrementally delivered as parts of an integrated process in the Software Integration Process.

Deployment View

The actual deployment of the A4Cloud Reference Architecture takes part in this view. WP:D-2 can suggest candidate deployment scenarios, listing the A4Cloud tools hardware requirements and placing the software modules in a deployment environment. This will also be enriched with a collaboration view of the real software modules/components/tools and their interactions. Since WP:D-2 delivers a Reference Architecture the deployment view is strongly dependent on the use cases to be implemented. The deployment view will ensure that Architecture Principle 7: and Architecture Principle 8: will be effectively addressed.

3.4 Selecting a proper Architecture Modelling Language

This section briefs on existing tools for modelling the accountability relationships and the components interactions to design the relevant architectural aspects and views mentioned above.

Business Process Modelling Notation

The standard Business Process Model and Notation (BPMN) [33] provides businesses with the capability of understanding their internal business procedures in a graphical notation and gives organizations the ability to communicate these procedures in a standard manner. Furthermore, the graphical notation facilitates the understanding of the performance collaborations and business transactions between the organizations.

BPMN is now widely used to enable business users effectively participating in the modelling of organizational and system processes. Such users can range from the business executives and analysts, who define the scope and the target outcome of the processes, to the system engineers and software developers, who are taking over towards implementing the technology that should facilitate the process envisaged functionalities. BPMN is also useful to those people that are responsible for managing and monitoring these processes and assess their proper execution.

This modelling language allows end-to-end business processes to be built. In these processes, many types of modelling can be covered, enabling the target outcome to be realized through a variety of dimensions, bringing together information and associated stakeholders. An end-to-end BPMN model can contain three basic types of sub-models: process, choreography and collaboration. By combining the three basic types of sub-models, a detailed representation of a business processes can be obtained.

Unified Modelling Language

The Unified Modelling Language (UML) is a standardised modelling language by OMG standardization body [34], which consists of an integrated set of diagrams, developed to help system and software developers accomplish a number of tasks towards implementing a software system. Such tasks involve the specification, the visualization, the architecture design, the construction and the simulation and testing of the system. UML was originally developed with the idea of promoting communication and productivity among the developers of object-oriented systems, but the readily apparent power of UML has caused it to make inroads into every type of system and software development.

Through UML, the functionalities, system and data models and processes can be represented, so that the software engineers can come up with a comprehensive architecture design, which includes Sequence and Component Diagrams. Sequence diagrams model the flow of logic within a system in a visual manner, enabling both to document and validate the logic behind the system's development, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artefact for dynamic modelling, which focuses on identifying the behaviour within a system. Other dynamic modelling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are the most popular design-level models for modern business application development.

Technical Architecture Modelling

The Technical Architecture Modelling (TAM) [35] extends UML and combines it with the Fundamental Modelling Concepts (FMC). It has been introduced by SAP, based on modelling best practices and a subset of diagram types, elements and notations used in UML. TAM distinguishes the architectural modelling at both a conceptual and design level. However, the adopted notation is the same for both levels. Thus, on the conceptual level, TAM adopts FMC and partially UML notation. When moving to the design regime, it uses activity diagrams, UML class diagrams and sequence diagrams from UML, while it adopts the block diagrams from the FMC notation.

A detailed analysis of the TAM notation and use is presented in [36].

Selection of Modelling Language for Architecture Design

The selection of one modelling language might be a major design step when developing a software system. The selection is mostly based on the capability of the available process modelling standardized technologies to meet the requirements set by the developers and the nature of the system itself. An indicative analysis of the capabilities of UML and BPMN notations is provided in [37] with respect to process modelling. When going to low level modelling, TAM and UML are the most dominant languages to be used, but, as explained in [38], UML fails in the modelling of the high level view of an architecture.

The selection of the A4Clolud Architecture Modelling Language has not yet been decided.

4 Instantiating the Reference Architecture

During the Software Construction Process, the actual implementation phase of a software product is taking place, which brings together software classes from the different layers of the Reference Architecture into physical A4Cloud tools. At this stage, fundamental implementation decisions must take place, which actually affect the results of the Software Integration Process. Due to this reason, this section tries to introduce the candidate technologies, tools and/or frameworks that should be considered in the different development areas and offer a baseline understanding among all partners on which of them are more suitable for the Software Construction and Integration Processes.

The Software Construction Process implements the various views of the A4Cloud Reference Architecture. Depending on the target layer of the architecture, specific decisions on the use of technologies are made. The contextual modelling of the Reference Architecture might also affect the selection process, mainly due to the fact that some functionalities could target to a group of stakeholders for which special attention is required.

The instantiation of the process view and the elaboration on the deployment view offer insight on how the Software Integration Process will be realised. Since this view refers to the interaction of the A4Cloud tools and their communication with optional external tools, the use of standards is highly recommended to maximise the interoperability potentials.

A general consideration for the selection of the appropriate technology is the fact that A4Cloud targets to open source software, which complies with the provisions of the Project Consortium Agreement. Thus, tools and technologies that are released under open source licenses with *Controlled License Terms*, like General Public License (GPL) and the Common Public License (CPL), which affect the proper Use and Dissemination of project results and their derivatives, should be avoided, unless no other apparent solution can be adopted.

The rest of this section approaches the integration process and fits the candidate technologies for the different areas of work. A summary of proposed technologies and standards to be used in A4Cloud is provided in Table 1.

4.1 Principles for integration

Cloud applications and services can be characterised from distributed data processing and storage functionalities. As such, the standardisation between the various design blocks in the cloud (i.e. application and infrastructure) for achieving interoperable interfaces to exchange meaningful information is necessary.

A4Cloud integration is loosely coupled. This means that the software deliveries of the A4Cloud tools are integrated in the context of cloud business processes, derived from the use cases in WP:B-3. In that respect, the use case logic remains independent, leading to both use case specific functional components and A4Clooud tools being self-contained and logically separated.

The integration approach will potentially follow a service oriented paradigm. This means that all the involved tools and software components maintain a well-defined input and output, which leaves the internal implementation agnostic to cloud actors. The implementation of a certain functionality may involve the execution of other services on the background, so that a cloud application can be realised through a composition of offered services. What should be considered though is the fact that any SOA-based protocols, which add overhead in the execution of distributed services, should be excluded from the cloud environment.

Following the development of the A4Cloud tools and their associated adopted tools to implement current accountability processes in the cloud service chain, the integration approach should deliver an incremental view of the accountability relationships among the cloud actors. This view will be based on well-defined and clearly described service interfaces for the various tools, which descriptions should be both human and machine readable, enabling developers to easily customise these tools and incorporate them as integral parts of their cloud service chain.

4.2 Cloud Data and Infrastructure Portability Interoperability Technologies

As per the considerations of the Open Group for the Cloud Computing Portability and Interoperability [39], *a system that involves cloud computing typically includes data, application, platform, and infrastructure components, where:*

- *Data is the machine-processable representation of information, held in computer storage.*
- *Applications are software programs that perform functions related to business problems.*
- *Platforms are programs that support the applications and perform generic functions that are not business-related.*
- *Infrastructure is a collection of physical computation, storage, and communication resources.*

In all these categories, portability and interoperability issues should be effectively addressed.

Data Access and Modelling Technologies

Data modelling technologies are extensively used in modern software development. The appropriate data structures are essential, so that integration of tools is sufficient and semantically correct.

Data management in cloud storage environment has been standardised, mainly through the Cloud Data Management Interface (CDMI) [40]. This standard has been proposed by SNIA and defines the functional interface that applications must use to create, retrieve, update and delete data elements from the cloud. As part of this interface the client will be able to discover the capabilities of the cloud storage offering and use this interface to manage containers and the data that is placed in them.

Extensible Markup Language (XML) [41] has been standardised as a data representation and exchange language by W3C. This language introduces a generic syntax, which can be realised both in human-readable and machine-readable forms. The machine-readable representation is exploited in software development to formalise data access, while human-readable representation is used by developers implementing such software.

Javascript Object Notation (JSON) [42] makes the data exchange and representation friendlier, by offering a text format, which is completely decoupled from the used software programming language.

Recently, Google introduced Protocol buffer [43], which is a language-neutral, platform-neutral, extensible mechanism for serializing structured data. This data representation and exchange format can structure data being generated from a variety of data streams and is interoperable with many programming language technologies.

Other data modelling languages include the Resource Description Framework (RDF) [44] and the Web Ontology Language (OWL) [45].

For the sake of the A4Cloud work, the use of JSON and protocol buffer data schemas are considered in the development of the A4Cloud tools in the different WPs of streams C and D, mainly because of their usability in defining close to human interpretation data representations.

Infrastructure Portability Technologies

In this paragraph, the existing cloud operating, service and infrastructure technologies are considered. In that respect, different available cloud computing platforms have been examined.

OpenStack [46] is an open source cloud computing environment. The extensive community support and the continuous growth in the adoption of market oriented solutions, along with the fact that it is an open source project have been considered as an asset for this selection.

OpenNebula [47] is another open source cloud computing solution, which enables building and managing enterprise clouds and virtualized data centers. However, an apparent decline in the popularity of this solution, raises concerns for the relevance of the platform as a basis for developing future-proof technologies in the short- to medium-term horizon.

SAP Hana Cloud [48] is a cloud platform developed by SAP. It consists of a Platform-as-a-Service cloud offering that comprises a number of SAP-developed added-value services to hosted web applications. In particular, it offers an access to the SAP HANA [49], an in-memory database with very high performance and many functionalities, particularly suitable for achieving real-time computing.

There are plenty of other cloud computing environments that could offer the necessary cloud testbed infrastructure for A4Cloud. A community based analysis of the most popular such solutions is given in [50]. In the context of the Project Horizons work in WP:A-2, the cloud providers landscape has been analysed [51][52][53][54]. Although a thorough analysis for the selection of one or another cloud environment is not the specific topic for discussion in this report, the available environments have been examined, based on specific criteria, such as their capability to host the envisaged functionalities of the A4Cloud tools, their maturity to offer state-of-the-art technologies, with good documentation and support, their popularity in the target markets and among the developers of cloud computing solutions (so that the potential risk on lack of required knowledge of the project developers to operate these environments can be mitigated) to leverage sustainability of the accountability developments and their applicability and flexibility in customizing their offerings to the business use case that will instantiate the Reference Architecture.

A4Cloud aims to provide a reference implementation of the Accountability Architecture that facilitates the needs of certain use cases arising from everyday business. As such, a selection of an appropriate cloud environment has to be made as a proof of concept. Of course, the Reference Architecture implementation shall satisfy Architecture Principle 5:. For the purposes of WP:D-7 work, OpenStack is going to be used as the appropriate cloud infrastructure environment both for the deployment of the cloud-based A4Cloud tools and the setup of the necessary cloud instance to act as an Infrastructure as a Service (IaaS) provider for the business scenarios to be developed. OpenStack is open source, popular and it can offer the necessary testbed for the instantiation of the A4Cloud Reference Architecture to the selected business user scenario(s).

OpenStack is very popular the last years and it is well placed among the top technologies for cloud offerings in the market, being one of the most promising open source cloud implementation solution for both a private or hybrid cloud. It is, actually, an emerging player in the cloud market [55] and many big technology providers worldwide stand as success stories on the use of OpenStack in real life problems, leveraging great opportunities for sustainable cloud support and offering huge potentials for further adoption in the future. The free availability of the infrastructure exhibits benefits, in terms of the solution to be able to be deployed at the customer site and be customised to the end user needs. Of course, OpenStack has its own drawbacks, mainly due to the lack of long standing maturity in the technology landscape, but it can eventually integrate with both legacy and third party technologies.

The extensive community support and the continuous growth in the adoption of market oriented solutions, along with the fact that it is an open source project have been considered as an asset for this selection. As shown in [46], “OpenStack HAVANA” is the eighth release of the open source software for building public, private, and hybrid clouds and has nearly 400 new features to support software development, managing data and application infrastructure at scale. Today, OpenStack is used for different application domains, like:

- Web / SaaS / e-Commerce, in which Cisco WebEx and PayPal being examples of this category OpenStack deployments for on-premise private cloud implementations.
- Academic / Research / Government, in which CERN, Harvard University, Hong Kong Cyberport Management Company Ltd and MIT being examples of this category OpenStack deployments for both on-premise private and public cloud implementations.
- IT, in which IBM and Intel, being examples of this category OpenStack deployments for various cloud implementations.
- Cloud hosting and Telco, in which HP Public Cloud and Rackspace are adopting OpenStack for open source cloud deployments.

OpenStack is being supported by a number of companies and organisations, including AT&T, Ubuntu, HP, IBM, Rackspace, Nebula, SUSE and Redhat being the platinum members of the foundation. An exhaustive list of supporters is included in [46]. Finally, the OpenStack community continues to attract the best developers and experts in their disciplines with 910 individuals employed by 145 different organizations contributing to the Havana release.

4.3 Communication and Interoperability Technologies

Application Interoperability Frameworks

Applications use specific standards to expose their derivatives to other systems and applications. In distributed systems, communication is achieved through APIs, which are implemented through various Web-based services.

Representational State Transfer (REST) [56] is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The term Representational State Transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Conforming to the REST constraints is referred to as being ‘RESTful’.

A RESTful web service (also called a RESTful web API) is a simple web service implemented using HTTP and the principles of REST. It is a collection of resources, with three defined aspects:

- The base URI for the web service, such as http://example.com/resources/
- The Internet media type of the data supported by the web service. This is often JSON or XML, but it can be any other valid Internet media type.
- The set of operations supported by the web service using HTTP methods (e.g., POST, GET, PUT or DELETE).

Unlike SOAP-based web services, there is no "official" standard for RESTful web services. This is because REST is an architecture, unlike SOAP, which is a protocol. Even though REST is not a standard, a RESTful implementation such as the Web can use standards like HTTP, URI, XML, etc.

RESTful web services are a key part of the “Web 2.0” momentum and are used by many applications and services. Developers prefer them over SOAP web services due to their simplicity and better performance. SOAP web services are used more often in enterprise application, usually in conjunction with other WS-* specifications that offer a formal model for security, workflow, notifications and other concepts.

SOAP [57], originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. SOAP exchanges messages in Extensible Mark-up Language (XML) format. It usually relies on other Application Layer protocols like Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP) for message negotiation and transmission.

A SOAP Web Service is formally described through a Web Services Description Language (WSDL) model, which is also based on XML. All major programming languages offer libraries tools for creating services and clients either from an existing WSDL definition (contract-first) or by code-first techniques.

SOAP is considered to be a part of WS-* specifications, which loosely refer to all specifications, associated with web services, and are maintained or supported by various standards bodies and entities.

One big advantage of SOAP over older technologies is that since it can use HTTP as the transfer layer, it can tunnel easily over existing firewalls and proxies, without modifications to the SOAP protocol, and over the existing infrastructure. The formal definition, which WSDL model offers, makes the interoperability between systems built with different technologies easier.

However, because of the verbose XML format, SOAP can be considerably slower than competing middleware technologies such as CORBA. This may not be an issue when only small messages are sent.

The OSGi framework [58] is a module system and service platform for the Java programming language. The OSGi framework tries to overcome Java's inherent modularity limitations and to implement a complete and dynamic component model.

OSGi can be used in order to:

- Ensure that code dependencies are satisfied before allowing the code to execute and thus avoid ClassNotFoundExceptions;
- Verify that the code dependencies are consistent with respect to required versions and other constraints’
- Easily share classes between modules;
- Package an application as logically indepenedent JAR files and be able to deploy only those pieces that are actually needed for a given installation;
- Declare which code is accessible from each JAR file and enforce the visibility (what is externally visible and what is not);
- Implement an extensibility (plug-in) mechanism;
- Start, stop, deploy, un-deploy and replace modules dynamically.

The OSGi Service Platform is composed of two parts: the OSGi framework and OSGi standard services. The framework is the runtime that implements and provides OSGi functionality. The standard services define reusable APIs for common tasks, such as Logging and Preferences.

The OSGi specifications for the framework and standard services are managed by the OSGi Alliance.

In A4Cloud, REST Web services will be used in order to enable interoperability among all the A4Cloud tools. They will eventually expose their own API and the appointed data streams will be conveyed as RESTful Web Services.

Communication Frameworks and Technologies

Apart from Web-based interfaces, integration focuses on the communication frameworks and technologies. Such technologies control the way that the offered services can be identified, discovered and consumed and the relevant information messages are being exchanged among various distributed components. Modern systems use complex communication frameworks aiming to provide wide integration solutions for a variety of disperse implementations.

The Enterprise Service Bus (ESB) [59] is a software architecture construct, which provides fundamental services for complex architectures via an event-driven and standards-based messaging engine (the bus). Developers typically implement an ESB using technologies found in a category of middleware infrastructure products, usually based on recognised standards. An ESB takes the complexity out of integration, providing connectivity to a wide range of technologies and creating services that can be reused across an organisation. An ESB does not itself implement SOA, but provides the features with which one may implement such. Developers can exploit the features of an ESB in order to integrate applications and services without custom code. Developers can shield services, regardless of location, from message formats and transport protocols. Data can be transformed and exchanged across varying formats and protocols.

Figure 3 depicts the architecture of a SOA system implemented with an ESB. The system integrates new and legacy applications that need to communicate with each other and exchange information. A variety of technologies, protocols and message formats are used. Thus, different programming and development languages can be supported and be linked together through ESB.

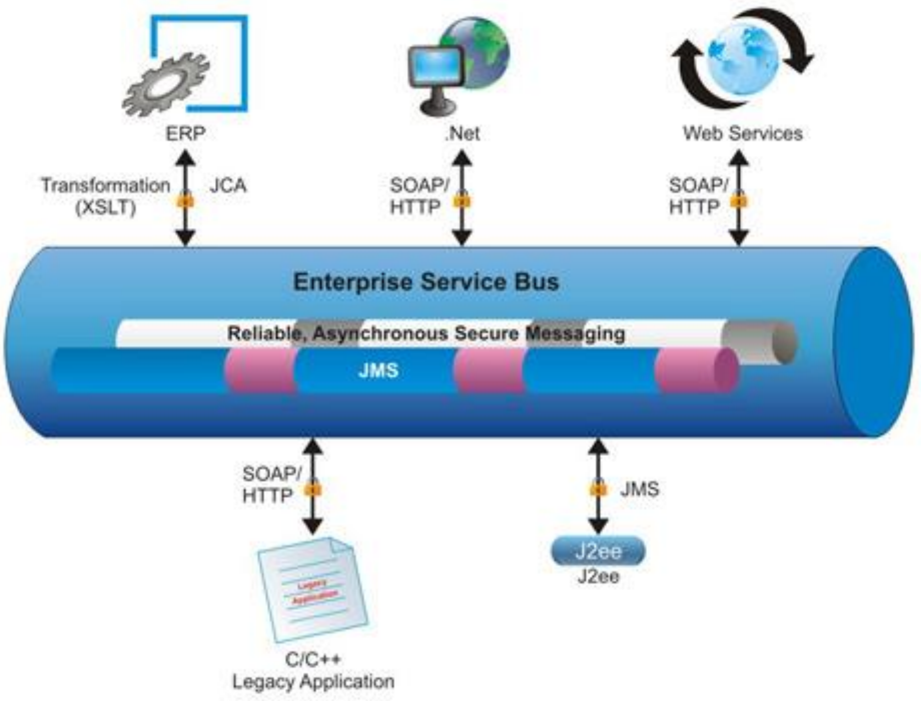

Figure 3: An example topology for ESB (please refer to [60])

The ESB hides the implementation details of one module from other components. All components send and receive messages to the ESB. The ESB is responsible for delivering the messages to the recipients in the appropriate format. If one component is momentarily unavailable or is being replaced by a new one, the system is not affected. The ESB waits for a component to be available again in order to deliver the message.

In A4Cloud, ESB is going to be used for message exchange and communication. Studying the available ESB implementations in the literature, a list of relevant solutions can be found, which exhibit different advantages and disadvantages for the project. The selection of one of these implementation is based on certain criteria, which are mainly lie on the assumptions that the project should adopt a reliable open source solution, which is widely used in other commercial and R&D activities and can be directly integrated into the project, without any major need for specialised customisation.

The most common open source ESB implementations and their corresponding license scheme are presented here:

- Mule ESB [61] is the most widespread used ESB system, which is released under the CPAL license (OSI-approved license, bundles other open source libraries that are made available under their respective licenses).

- Apache Service Mix [62] is released under Apache license and includes a Java Business Integration container to support enterprise strength solutions.
- Swordfish [63] is another ESB solution provided under the Eclipse Foundation Software User Agreement. The project is in incubation phase, which is the main drawback for using it in commercially strength solutions.

Given these solutions, among other existing ones, the Mule ESB and Apache Service Mix are the most promising ones and they have been selected in A4Cloud to support the communication between the different tools. The selection has been based on the following factors:

- Both Mule ESB and Apache Service Mix are open source popular solutions for an ESB implementation;
- They have extensible documentation and support communities in case the project faces any problems importing them into A4Cloud;
- The exploitation of two popular ESB solutions can maximise the acceptance and independence of the A4Cloud developments to specific ESB implementations.

Most probably, only MuleESB will be used in A4Cloud.

4.4 Software Development Technologies

This section analyses some commonly used software development environments for cross language programming. It focuses on the Integrated Development Environments (IDEs) and based on the preliminary study and comparison listed in [64], it presents the Eclipse platform, including the associated plugin development, and the NetBeans Platform. It must be highlighted that these IDEs do not restrict the development programming language, as both Eclipse and NetBeans can work with many languages, including Java, C++, Go, Perl, Tcl and Python.

Eclipse Platform and Eclipse plug-ins

The Eclipse Platform [65] is a multi-language software integrated development environment (IDE), which mostly refers to Java developers, but offers plug-ins to support other programming languages. Eclipse employs plug-ins [66] in order to provide all of its functionality on top of (and including) the runtime system, in contrast to some other applications where functionality is typically hard coded. The runtime system of Eclipse is based on Equinox [67], an OSGi [68] standard compliant implementation.

This plug-in mechanism is a lightweight software framework. In addition to allowing Eclipse to be extended using other programming languages, such as C and Python, the plug-in framework allows Eclipse to work with typesetting languages like LaTeX, networking applications such as telnet, and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and Concurrent Versions System (CVS) support is provided in the Eclipse Software Development Kit (SDK), with Subversion support provided by third-party plug-ins.

With the exception of a small run-time kernel, everything in Eclipse is a plug-in. This means that every developed plug-in integrates with Eclipse in exactly the same way as other plug-ins; in that respect, all features are "created equal". Eclipse provides plug-ins for a wide variety of features, some of which are through third parties using both free and commercial models. Examples of plug-ins include a UML plug-in for Sequence and other UML diagrams [34], a plug-in for Database Explorer, and many others.

As stated above, Eclipse is an open platform, being designed to be easily extensible by third parties. The Eclipse SDK consists of the following layers, as shown in Figure 4 and [69]:

- The Rich Client Platform (RCP): On the bottom is RCP which provides the architecture and framework to build any rich client application.
- The Integrated Development Environment (IDE): It is a tools platform and a rich client application itself. We can build various form of tooling by using IDE for example Database tooling.
- The Java Development Tools (JDT): It is a complete java IDE and a platform in itself.

- The Plug-in Development Environment (PDE): It provides all tools necessary to develop plug-ins and RCP applications.

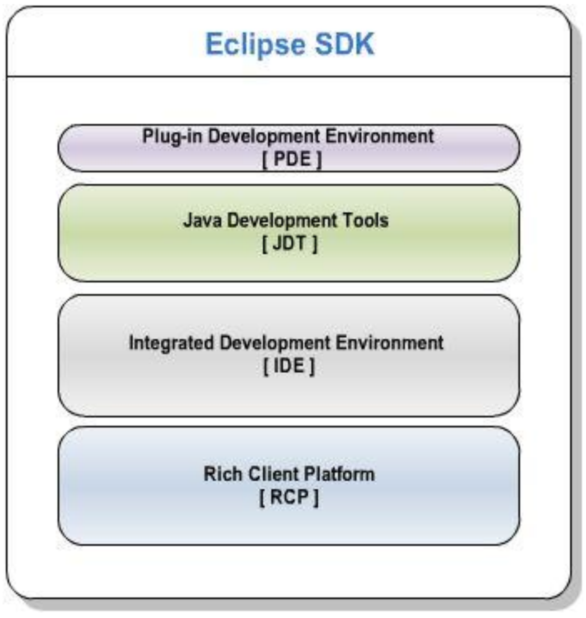
Eclipse SDK
Plug-in Development Environment [PDE]
Java Development Tools [JDT]
Integrated Development Environment [IDE]
Rich Client Platform [RCP]

Figure 4: The Eclipse plug-in layers (please refer to [69])

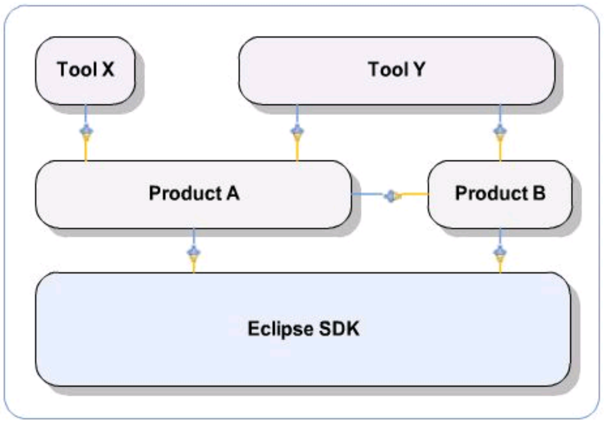

Figure 5: The Eclipse plug-in architecture (please refer to [69])

All the layers in eclipse SDK are made up of plug-ins. Except from a small kernel, almost all of the functionality of eclipse is located in different plug-ins.

Around the Eclipse SDK, new products and tools (plug-ins) can be created, as shown in the Eclipse plug-in architecture in Figure 5 and [69]. A plug-in is nothing but another java program, which extends the functionality of Eclipse in some way. Each Eclipse plug-in can either consume services provided by other plug-in or can extend its functionality to be consumed by other plug-ins. The plug-ins are dynamically loaded by Eclipse at run time on demand basis.

A plug-in can be delivered as a jar file. A plug-in is self-contained bundle in a sense that it contains the code and resources that it needs to run: code, image files, resource bundles etc. A plug-in is self-describing by means of describing which it is and what it contributes to the world. It also declares what it requires from the world.

NetBeans

NetBeans [70] is an open-source project dedicated to providing rock solid software development products (the NetBeans IDE and the NetBeans Platform) that address the needs of developers, users and the businesses who rely on NetBeans as a basis for their products; particularly, to enable them to develop these products quickly, efficiently and easily by leveraging the strengths of the Java platform and other relevant industry standards.

As an IDE, NetBeans is considered to be the original free Java IDE. However, it provides support for several other programming languages, like PHP, JavaFX, C/C++, JavaScript, etc. and frameworks. It has a large community of users, who can benefit from the support for fast and smart code editing and the bug free coding, which is enhanced with the NetBeans profiler to provide expert assistance for optimizing the speed and the memory usage of applications, while it makes it easier to build reliable and scalable applications.

Selecting Development Programming Language

No restrictions are applied on the selection of the programming language that will be used in the implementation of the A4Cloud tools. However, in order to minimise the tool dependencies on alternative operating systems and software environments, Java is highly recommended as programming language for tool development. Some implementation in WP:C-7 and WP:C-8 will be, also, based on the Go Project [71], which has been introduced by Google, as an open source alternative for software development. Go is structured into packages and comprises an expressive, concurrent, garbage-collected programming language.

As it will be presented later in the document, in order to ensure the smooth interoperability between the various tools and their integration in the context of business applications, every tool must be designed, so that it exposes machine readable application programming interfaces (APIs) in HTTP RESTful format. It is, finally, suggested that the A4Cloud tools will be built and executed at the runtime environment (e.g. Ubuntu Linux) with the necessary tools needed in each case.

4.5 Security Technologies

This section summarises the security technologies, which are considered in the project Software Construction and Integration Processes. The areas that A4Cloud will work include the frameworks for identity management services, the languages for policy specification and management, the secure exchange of information and communication among the physical nodes of the A4Cloud set up environment and the tools for secure monitoring and incidence response, as part of the implementation of corrective mechanisms towards preserving privacy, data protection and accountability.

Identity Management

Identity Management facilitates the functionalities for policy-based authentication and/or authorisation. There are plenty of custom and standardised technologies to serve authentication and authorisation, which have been extensively used and tested in different scale level systems. The main difference between these technologies are on whether the actions are monitored by a local or a third-party service, which is attributed the task to perform identity management functionalities and communicate the results back to the specific system/application. The primary scope for the use of third party services for authentication and authorisation is the achievement of single-sign-on solutions. And since interoperability is by principle a major target for modern ICT systems, it is frequent nowadays to follow such solutions. Of the most popular frameworks are the OpenID and the OAuth, which are described in the following.

OpenID [72] facilitates for single-sign-on by providing mechanisms for both authentication and authorization processes. Both request and response message formats are defined to facilitate the transmission of necessary credentials within a Web service activity. It is decentralized standard, meaning that it is not controlled by any website or service provider.

OpenID allows use of an existing account to sign in to multiple websites, without needing to create new passwords. It requires a Web Application to be OpenID-enabled, meaning that the authentication process can be achieved through an external service provider.

Open Authorisation (OAuth) [73] is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications. OAuth allows users to share their private resources stored on one site with another site without having to hand out their credentials, typically username and password. This is achieved through tokens, exchanged between service providers. Each token grants access to a specific site for specific resources and for a defined duration. This allows a user grant a third party site access to their information stored with another service provider, without sharing their access permissions or the full extent of their data.

Currently, OAuth is exposed on the Web through three versions, namely OAuth 1.0, 1.0a and 2.0. OAuth 2.0 is the next evolution of the OAuth protocol and is not backward compatible with OAuth 1.0. The OAuth 2.0 authorization framework is an IETF standard for authorization, which enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

It should be noted that OAuth is distinct to OpenID and, thus, differentiates from it by complementing the process of authentication and authorization is software and service distributed environments. The need for one or both the solutions is not a de facto case in A4Cloud. Currently, there is no clear clue on whether we have to go with such Identity Management services, but it is highly considered.

Policy Management

Several policy representation languages and approaches has been proposed in recent years [74][75][76][77][78][79]. In this section, we briefly introduce XACML [74] the de facto standard for access control policies representation and PPL [75], an extension of the XACML language, combining access and data usage policies.

XACML stands for eXtensible Access Control Markup Language and allows policy and access control requests/responses description. The authorization model introduced by the XACML specification is illustrated in Figure 6. When a client, i.e. data requester, wants to take some action on a resource he first makes a request to a policy enforcement point (PEP) protecting the resource. If the request is not written in XACML, the PEP maps it to an XACML request and forwarded to a policy decision point (PDP). The PDP reads the request, search for a policy that applies to it and evaluates the request according to that policy. The decision is then written in XACML and sent to the PEP that informs the client.

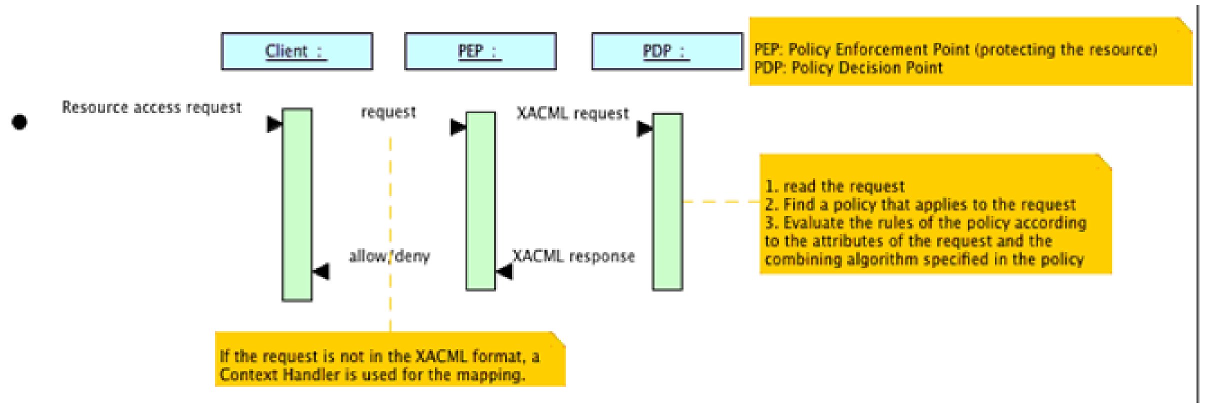
Client :
PEP :
PDP :
PEP: Policy Enforcement Point (protecting the resource)
PDP: Policy Decision Point
Resource access request
request
XACML request
allow/deny
XACML response
1. read the request
2. Find a policy that applies to the request
3. Evaluate the rules of the policy according to the attributes of the request and the combining algorithm specified in the policy
If the request is not in the XACML format, a Context Handler is used for the mapping.

Figure 6 Simplified XACML authorization model

An advantage of XACML is that it presents many extensible points that can serve to express accountability related properties. However, it lacks support for privacy and usage control. See [80] for a detailed study of XACML and its inadequacy for the A4Cloud context.

PPL, PrimeLife Policy Language, extends XACML with privacy-enhancing, credential-based and usage control features. PPL inherits the XACML properties and authorization model and extends them to support some extra accountability related properties (usage control and logging[1] for instance). However some features, like auditing for instance, are not supported.

Among the existing policy representation languages, PPL is the best candidate for policy representation in the A4Cloud context as it provides different features for accountability properties representation [80]. Hence, PPL can be extended and used as an accountable policy language in A4Cloud (for that reason, we call it APPL). In this context, in Stream C, and particularly in WP:C-4, we are extending PPL to provide an accountable policy language that meets the A4Cloud needs.

[1] Logging is only partially supported in PPL. The current specification allows to declare the action to log, but the conditions of the logging cannot be specified.

Secure Communication

Transport Layer Security (TLS) [81] is a cryptographic protocol aiming to provide secure communication over the Internet. TLS is an upgrade of SSL version 3.0. It offers privacy between communicating applications by ensuring that no third party will tamper with any exchanged message. TLS is composed of two layers: TLS handshake and TLS record. TLS record protocol is layered on top of TCP and is responsible for assuring privacy by applying data encryption. TLS handshake protocol acts as the mechanism to authenticate each member of a connection and to initiate the encryption algorithm and the cryptographic keys, before data is transmitted or received. This protocol is highly considered in A4Cloud.

In the scope of A4Cloud, if needed, we could use a Certificate Authority and a Public Key Infrastructure to verify the relation between a certificate and its owner, as well as to generate, sign, and administer the validity of certificates. A crypto module will implement the security mechanisms needed for the communication between the modules of the platform.

Monitoring and Incident Response

Since the first appearance of cloud computing as a distributed technology, it became imperative that there should be a way for monitoring its various components. As described in the reference architecture from NIST [82], its constituent parts need to be monitored at all times in order to provide both cloud customers and providers with real-time information about the system's state. This task of monitoring cloud-based systems can be very challenging for a number of different reasons. The perplexity of cloud infrastructures, the great degree of dependability in-between the layers of the cloud stack and the diversity of the usage of the analysis of monitoring data to accommodate different objectives, set the definition and design of a single conceptual framework a rather difficult task.

The process of monitoring can be the ears and the eyes that make it possible for cloud customers, cloud brokers and cloud clients to become aware of the cloud's condition. Normally it involves the collection of events that take place in the cloud environment with the use of event captors that are later on analysed and further assessed. That information can be data such as system metrics that can be obtained directly from the monitored system (CPU, memory usage, number of IOs) or can be more complicated ones that require calculations to be done (percentage of successful requests over a specific period of time, availability of a service between certain time intervals, etc.). For those cases a series of commercial and open source tools have been developed over the course of time, with Nagios [83] and Ganglia [84] being probably the most well-known ones. Also, ISP providers in some cases offer proprietary software or monitoring services that are specific to the cloud services that they offer. In that category fall CloudWatch [85] from Amazon and RackSpace Cloud Monitoring [86] from Rackspace used respectively to monitor cloud resources. All those monitoring tools collect system information while the system is up and running and allows administrators to define rules that if violated, specific actions can take place. Most commonly this type of monitors are used to perform health check operations of the cloud environment and produce reports about the system's overall behaviour and performance.

Despite the commonly held perception that monitors are just event collectors, as far as distributed environments like cloud platforms is concerned, they are becoming more and more complicated systems. They are intended to be used for more sophisticated operations than just an alerting mechanism if e.g. CPU goes above a specific threshold or if IOs become slower than a lower bound. In cloud computing part of the responsibility for the applications has been abstracted away from the providers taking the burden of the system and application maintenance from cloud clients. However, this comes with a price which is that clients are obliged to give away a great deal of the control that they used to have over their own systems and applications. The lack of control and the fact that in reality they are agnostic or have very little insight of the internal operations that take place within their providers environment, are the sources for a series of concerns with respect to the security and privacy of their cloud applications and data that monitoring aspires to overcome. Several implementations of monitors have been built [87][88][89][90] like MONIT [91] and EVEREST [88] trying to address the problem of the monitoring of security properties by providing a monitoring a framework for that purpose. In the aforementioned monitors a number of security properties are being examined such as confidentiality, integrity and the application of access controls at the different layers of the cloud stack.

As a complement to the previously mentioned benefits gained from monitoring, we need to also include cloud service and infrastructure certification. A monitoring based certification framework with the name CUMULUS [92] is an under development FP7 project that aspires to provide a set of tools and semantics that will allow the certification of cloud environments in a formal and rigorous fashion. Also, continuous monitoring can be helpful for enhancing accountability in the context of the relationship between cloud providers and customers. As mentioned in [93] monitoring is a contributing factor that if combined with risk analysis, policy enforcement and compliance auditing can help us address the issue of accountability and use monitoring as a means for the collection of as indisputable as possible evidence, in order to hold accountable the proper entity when the system does not behave as expected. This implies that SLAs need to be well defined, that they contain no ambiguities and that the monitors that gather system information have to hold the property of non-repudiation.

As mentioned before in this section, we consider monitors a set of components that need to seamlessly collaborate in order to capture, analyse and eventually provide us with some sort of analysis and diagnosis on the collected data. As a consequence, the data that is gathered can be used in order to conduct incident management over the systems that are being monitored. That means that we can use the events captured to enable automatic responsiveness of the system when it does not operate normally in an attempt to rectify potential errors and restore normal operation. Most of the research that has taken place in this area has mainly focused on the definition of a set of guidelines and best practices [94], as well as on what is known as adaptive security mechanisms.

Those incident management systems respond to incidents by taking into account a measured security level [95][96], something that effectively improves the system’s response to potential incidents. An example of such a system comes from the domain of health Clouds [97], adaptive authentication [98] and automated incident management for PaaS [99]. As a general comment, incident management tools are automated systems that are able to identify deviations of the system’s normal operation, as described in a formal way in SLAs, and attempts to take simple corrective actions in order to bring the system back to its original condition. Those corrective actions in some cases can be simple steps such as to restart an application or restart a service. However if those simple actions taken won’t bring any results it becomes necessary for more radical operations such as rebooting the system or reimaging of the virtual machine where the application is hosted.

It must be noted that the selection of standardised technologies for monitoring and incidence response is subject to the subsequent selection of the underlying cloud infrastructure environment, as presented in Section 4.2. This will be further examined in WP:C-8.

4.6 Development of User Interfaces

When developing User Interfaces (UIs) for the A4Cloud project, a major concern is transparency awareness. The technologies that are used for the interaction of the end users with a software solution are strongly coupled with the available end user devices and the purpose of the envisaged interaction. Currently, a plethora of available devices are being used to enable users interact with software and systems from intuitive UIs. Such end user machines range from Web servers and desktop PCs to mobile devices and tablets and they, subsequently, expose different capabilities, which should be seriously considered by the respective UI developers. The developers may, thus, make fundamental UI design and development decisions on the adoption of one over another programming language for the UI development.

Another criteria that should be taken into consideration is the scope of interaction of the end user with the software. Whether the user has to consume an online service or to experience with local software to manipulate digital data is of major importance as well. New trends in software and services evangelise the connected Future Internet and the interoperable systems, which has been widely spread with the invasion of interactive Web 2.0 technologies. The need for minimising the dependency on software installations, limiting the access restrictions from anywhere and enabling the online collaboration have constituted these technologies dominant in the UI development area. However, still much of software is oriented to native application programming (desktop products and solutions, native mobile application development, etc.), especially when the coupling to low level Operating System (OS) functionalities is strong.

Regarding technologies, depending on whether the user interface is a desktop or a Web-based application, a variety of programming languages can be used. The main criteria have to do with the fact that the technologies should be easy, user friendly and be supported by an active community, so that maintenance capabilities are maximised. Native compilation can be another restriction.

For desktop UI applications, standards include Java, .NET, Go etc. When going to Web-based UI applications, the selection of a proper scripting language or combination of them is of outmost importance. For example, HTML5, PHP, JavaScript, Go, etc. are used to express the UI functionalities to be implemented for the end user. The question, however, has to do with the privacy issues that the UI development should respect in order to minimise the execution of the client UI application to external to end user dependencies.

The purpose of use might result to either a desktop of Web-based UI application, so a unique recommendation on the appropriate use of UI development technologies cannot be made. When this is appropriate, A4Cloud will target to Web-based UIs to minimise the dependency to local operating systems and environments. However, in all cases, A4Cloud should consider for transparent aware and privacy protected User Interfaces. This is the criterion on the most suitable technology to be used in the end.

4.7 Application Server Technologies

The selection of Web-based UIs has direct impact on the use of application server technologies to build the server side functionalities of Web-based applications. A number of frameworks for application server technologies exist that are analysed in the following.

Oracle Glassfish Application Server

GlassFish [100] is an open source application server project led by Sun Microsystems (owned by Oracle corporation) for the Java EE platform. The proprietary version is called Sun GlassFish Enterprise Server. GlassFish is free software, dual-licensed under two free software licenses: the Common Development and Distribution License (CDDL) and the GNU General Public License (GPL) with the classpath exception.

The newest version of the server is GlassFish 3.1.2.2, which is based on world’s first implementation of Java EE 6 with an OSGi based flexible, lightweight, extensible platform. It requires a small memory footprint. It is fully featured with production-ready features such as clustering and high availability provides optimized runtime performance and ready for enterprise deployments.

JBoss

JBoss Aplication Server or WildFly [101] is an open source solution for application server development supported by Red Hat. With JBoss, a better way to build applications is offered, using modern technologies, with a recommended approach that makes developers to be more productive. JBoss builds upon Java EE 6 and CDI, enabling the deployment of applications locally or to the cloud.

Apache Felix

Apache Felix [102] is a community effort to implement the OSGi R4 Service Platform and other interesting OSGi-related technologies under the Apache license. Felix and Equinox are considered to be the reference implementations of OSGi specifications.

The Felix project is organized into subprojects, where each subproject targets a specific OSGi specification or OSGi-related technology. Many enterprise application servers, like Glassfish, ServiceMix and JOnAs use Felix internally.

Apache Tomcat

Apache Tomcat 6 [103] is a web server that also implements JSP and Java Servlets technologies. It will be used for serving the web UI of the platform and any other module that need to be exposed as web services; for the moment there is no such a plan but in the future there might emerge new needs for the platform.

Java Agent DEvelopment Framework (JADE)

The Java Agent DEvelopment Framework (JADE) [104] is a framework for Intelligent Agents that can assist humans and act on behalf of them to perform certain automatic and repetitive activities and execute task, which need to learn from the environment.

JADE simplifies the implementation of intelligent agents. This multi-agent system complies with the FIPA specification and is licensed under the terms of LGPL (Lesser General Public Licence Version 2). The

Foundation for Intelligent Physical Agents (FIPA) specification supports coordination between agents and provides a standard communication language between agents.

JADE can be distributed across several machines, enabling communication encryption, agent remote control and the capability to merge agents between platforms. All aspects that are not agent application dependent, such as message transportation, message encoding/decoding/parsing, or agent life cycle management are provided by the platform. JADE also provides the developer with graphical tools for controlling, deploying and debugging agents.

Selection of Application Server Technologies

Due to the planned development of the A4Cloud tools, the use of Application Server Technologies is seen as necessary. The current analysis of tools shows that different components have different requirements, mainly because they can base their development on existing background knowledge or they foresee on the use of specialised implementation. On the other hand, the project will consider minimising the use of different application server technologies to eliminate the dependencies when deploying different components in the A4Cloud set up environment. As stands now, the majority of the tools will be based on Apache Tomcat, while a couple of tools will use JBoss and JADE. The use of JADE might introduce the need for the implementation of a broker software component that will wrap the intelligent agent functionality and communication protocol in standardised Java objects and expose the appropriate interfaces as Web Services.

4.8 Data Processing Technologies

Integration follows different approaches depending on the nature of the project, the methodology that is adopted and the degree of dependencies among the different software products that need to work together. A major issue is the compliance of all developments in the selected software development approach to common strategic decisions that make the loosely-coupled integration and the software quality assurance processes to run smoothly.

At the initial stages of this WP:D-2, the need for a Continuous Integration (CI) process had been examined. CI consists of practices, such as daily builds and additional checks, which target to avoid the appearance of common defects to the extent this possible. In order to enable automatic daily builds, CI software gathers the whole source in one place (with different revisions), automates the build process and testing, and provides the latest working executable to anyone involved in the project. The CI model performs a set of activities for the process implementation: building the system, running tests, deployment activities, and finally reporting test and deployment results.

The CI practice assumes a high degree of tests which are automated into the software: a facility that can be seen as “self-testing code”, often using a testing framework, such as JUnit4 [105] in case of Java development. Each automated build will perform the following steps:

- Compilation and creation of the build package;
- Creation of a report on code metrics (such as package, dependency analysis and cyclomatic complexity);
- Creation of a report on how much source code follows declared Coding Standard;
- Creation of a report on possible bugs by a static code analysis execution;
- Automated deployment in testing environment;
- Execution of automated test cases and creation of test report;
- Publishing the build artifacts;
- Notification to stakeholders and involved developers about build outcome by email and on central build dashboard;

CI is supported by a number of tools, like Hudson [106] or Jenkins [107] for monitoring the status of a Source Control System (SCS) and making builds, based on changes occurred. Builds are realized through tools, like Apache Maven [108] and Apache Ant [109], and are controlled in terms of their inter-dependencies from advanced repository management tools, like Artifactory [110]. The building process is supported by issue tracking and ticketing systems, like Trac [111] and Redmine [112], while quality control and check is considered through frameworks, like JUnit [105] or Sonar [113].

As suggested above, A4Cloud follows a loosely-coupled approach for the integration, in which the different cloud-based or distributed tools expose RESTful Web interfaces, whatever programming language has been used for their implementation (but using java wrappers, whether this is necessary). As stands now, it seems that there is no clear need for using a centralized tool for building and managing the artifacts of each module by following a CI approach. On the contrary, the variety of different technologies used for each tool would make a continuous integration process setup extremely complex and actually without real benefit, since all tools can act as loosely coupled, standalone processes.

In that respect, the instant snapshot of the A4Cloud Reference Architecture to integrate accountability-based practices to current business processes can be more flexible. However, the different A4Cloud tools will make use of the common framework for optimising the data processing needed in the support of the accountability attributed in large scale environments. Thus, the process and deployment views of the A4Cloud Reference Architecture will identify the necessary integration points and the members of different teams involved in these points will work closely together to solve the dependency problems along the A4Cloud tools interactions.

This ad-hoc approach will be supported by two main common spaces of work, namely the data storage layer technologies and the code versioning system technologies, which are analysed in the following sections. Before that, we present the selected candidate technologies for large scale data processing.

Large scale data processing may be needed in A4Cloud for the collection and analysis of the logs from the use of personal data stewardship in the cloud. Distributed file sharing and storage will be implemented through Hadoop [114] and Hadoop Distributed File System (HDFS). Hadoop is an open source software framework initiated and maintained by the Apache Software Foundation and is licensed under the Apache License 2.0. Hadoop enables the distributed and large scale processing of big data sets across clusters and is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance.

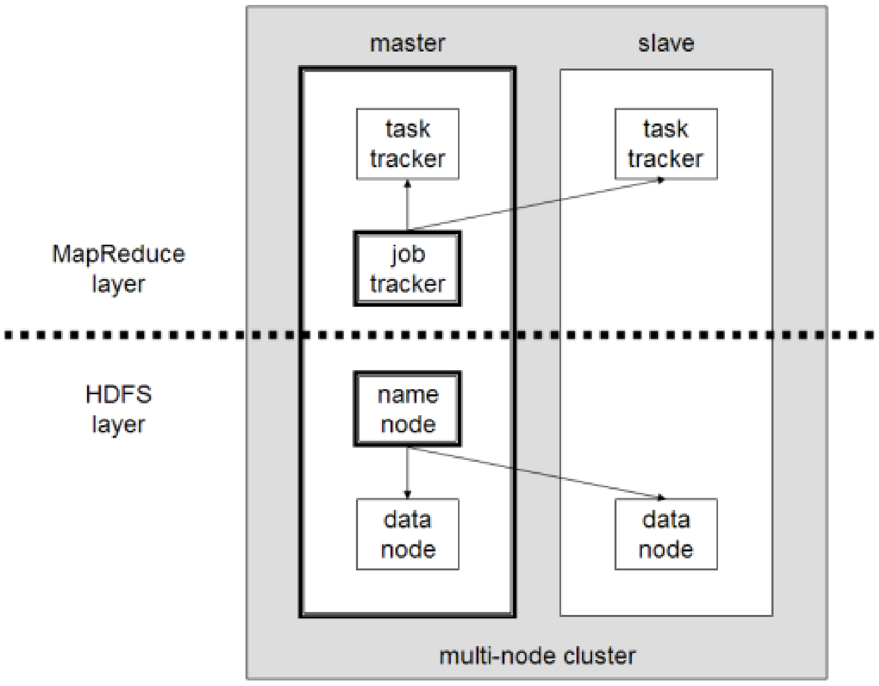

Figure 7: An example Hadoop cluster (see [114])

The Hadoop framework is mainly composed of the following modules [114]:

- Hadoop Common which contains libraries and utilities needed by other Hadoop modules
- HDFS, which is a distributed file-system that stores data on the commodity machines

- Hadoop YARN, a framework for job scheduling and cluster resource management.
- Hadoop MapReduce, which is a programming model for large-scale data processing.

Hadoop is supplemented by an ecosystem of Apache projects, such as Mahout, Pig and Hive, etc., which extend the value of Hadoop and improve its usability.

In Hadoop installations, there are four main concepts, NameNodes , DataNodes, JobTrackers and TaskTrackers (see Figure 7). From the HDFS perspective the NameNode keeps track of where the data in the cluster are located, thus, basically it holds a list of the DataNodes and what data they hold. The data node is where the actual data resides. From the MapReduce perspective the JobTrackers act as the mediator between the application and the MapReduce engines and handles the resource management of the engine, issuing jobs to the TaskTracker. One cluster node can have at the same time several different roles

A small Hadoop cluster includes a single master and multiple worker nodes, as shown in Figure 7.

HDFS complements Hadoop by providing a distributed file system, meaning that it spreads storage across multiple nodes. The key features are [115]:

- HDFS stores files in blocks typically at least 64 MB in size, much larger than most file systems.
- HDFS is optimized for throughput over latency
- HDFS is optimized for workloads that are generally of the write-once and read-many type.

Instead of handling disk failures and need RAID controls, HDFS uses replication. The data are stored on multiple nodes within the cluster and the NameNode constantly monitors failure reports sent by each DataNode. If these reports indicate that data have fallen under a certain replication threshold, it schedules the addition of another copy within the cluster.

Along with HDFS, MapReduce is a processing paradigm that provides a series of transformations from a source to a result big data set. In the simplest case, the input data is fed to the map function and the resultant temporary data to a reduce function. The developer only defines the data transformations. Another large scale data processing and analysis environment consists of RHIPE, the integration of Hadoop with R language [116]. This will be used for the evidence processing in WP:C-8, following the R2Time framework [117].

The MapReduce engine in Hadoop manages how these transformations are applied to the data across the cluster in a parallel way. The data are provided to the map function as a series of key value pairs. The output of the map function is a set of other key value pairs, and the reduce function performs aggregation to collect the final set of results.

Hadoop provides a standard specification (that is, interface) for the map and reduce functions, and implementations of these are often referred to as mappers and reducers.

4.9 Persistency and Data Storage Technologies

This section presents the available standardised technologies for persistency and data storage. Storage can be achieved by utilising structural or unstructured data forms.

MySQL [118] is considered to be the most popular object relational database system. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation.

Free-software-open source projects that require a full-featured database management system often use MySQL. MySQL is the M in the LAMP acronym, which stands for Linux-Apache-MySQL-PHP and refers to the open source technologies used by numerous frameworks and applications.

MySQL works on many different system platforms, including AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, Mac OS X, Microsoft Windows, NetBSD, Novell NetWare, OpenBSD, OpenSolaris and others. It is a very mature system, very good documented and with a wealth of resources available. Many graphical interface applications for the management and administration of a MySQL database are available, both free and paid.

Although MySQL was the database of choice for almost all open source projects, the acquisition of Sun by Oracle, which brought MySQL under Oracle's control, made many developers sceptical about using it in new projects. Although Oracle promised to continue supporting the database system and keep offering the community edition, it is generally thought that a sudden policy change from Oracle's part is not to be excluded. For this reason PostgreSQL is considered as an alternative for open source projects.

PostgreSQL [119] bypasses the GPL limitation and is released under a liberal open source license, the PostgreSQL License (MIT-style license). It is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL:2008 data types. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, among others, and exceptional documentation.

PostgreSQL SQL implementation strongly conforms to the ANSI-SQL:2008 standard. It has full support for subqueries (including subselects in the FROM clause), read-committed and serializable transaction isolation levels. And while PostgreSQL has a fully relational system catalogue, which itself supports multiple schemas per database, its catalogue is also accessible through the Information Schema as defined in the SQL standard.

As with many other open-source programs, PostgreSQL is not controlled by any single company — a global community of developers and companies develops the system.

MongoDB [120] is a popular NoSQL database solution. NoSQL refers to a broad class of database management systems that differ from classic relational database management systems (RDBMSes) in some significant way. These data stores may not require fixed table schemas, usually avoid join operations and typically scale horizontally. NoSQL databases provide no SQL interface and usually rely on much simpler interfaces that use associate arrays or key-value pairs. Besides their simplicity, a big advantage of many NoSQL technologies is that they use a distributed architecture, which allows them to easily be deployed in a cloud system. A NoSQL system can be deployed in many servers and a failure of a server can be tolerated.

MongoDB database is document-oriented and manages a collection of JSON-like documents. MongoDB can run in almost all operating systems and binaries are available for Windows, Linux, OS X and Solaris. MongoDB uses JavaScript for making queries. It offers official drivers for all popular programming languages (C, C++, C#, Haskell, Java, Javascript, Perl, PHP, Python, Ruby, Scala).

HBase [121] is Hadoop’s database, which provides a platform for storing and retrieving data with random access from a big data store and runs on top of HTFS. HBase can store structured and semistructured data. It can also store unstructured data too, as long as they are not too large. HBase is designed to run on a cluster of computers instead of a single computer. It isn’t a relational database so it doesn’t use SQL to make queries to the data and it doesn’t enforce relations between its database objects. Like HTFS and MapReduce, HBase has a master node that is responsible for the management of the cluster and a number of distributed nodes that do the actual work with the data.

Unlike traditional relational databases that require structured data with well-defined schemas, MapReduce and Hadoop work best on semi-structured or unstructured data. On top of that, the need for distributed or local storage is driven by the analysis of the business use case, which will instantiate the Reference Architecture. In that respect, the project will examine the requirements for relational or not data structures, considering the fact that HBase might be the appropriate candidate technology for cloud storage, while in some cases a special HBase implementation, called OpenTSDB [122], will be considered.

4.10 Code Versioning Solutions

In order to support integration, A4Cloud targets to the use of a code versioning solution. Relevant available tools include Subversion [123] and Git [124], which are briefly presented in the following.

Subversion is a software versioning source code and a revision control system developed by the Apache Software Foundation and distributed under free license alternatives. Features include commitments as true atomic operations (interrupted commit operations in CVS would cause repository inconsistency or corruption). The tool offers full revision history for management functions, like rename, copy, move and remove of files. It maintains versioning for directories, renames, and file metadata, in which users can move and/or copy entire directory-trees very quickly, while retaining full revision history. It can integrate with Apache HTTP Server as network server, using WebDAV/Delta-V for protocol. There is also an independent server process called svnserve that uses a custom protocol over TCP/IP.

The tool is client-server, which follows a layered library design to send diffs in both directions. The output can be parsed in different formats, including XML log output. The bindings support many programming languages, such as C#, PHP, Python, Perl, Ruby, and Java.

Git is another free and open source distributed version control system. It is more efficient than more similar tools like Subversion by offering additional features like cheap local branching, convenient staging areas, and multiple workflows. Every Git working directory is a full-fledged repository with complete history and full version tracking capabilities, not dependent on network access or a central server.

In A4Cloud Git will be considered as the code versioning solution. The benefits of using Git over Subversion are listed here:

- Git is much faster than Subversion
- Git's repositories are much smaller than Subversions (for the Mozilla project, 30x smaller)
- Git was designed to be fully distributed from the start, allowing each developer to have full local control
- Subversion allows you to check out just a subtree of a repository; Git requires you to clone the entire repository (including history) and create a working copy that mirrors at least a subset of the items under version control. However, Git branches are simpler and less resource heavy than Subversion's and they can carry their entire history
- Git provides better auditing of branch and merge events
- Git's repo file formats are simple, so repair is easy and corruption is rare.
- Git repository clones act as full repository backups. Backing up Subversion repositories centrally is potentially simpler.

4.11 Guidelines on Release Management and Software Documentation

Using the code versioning proposal for the A4Cloud implementations shown in section 4.10, a release management approach will be established in the Reference Architecture documentation to ensure that the different implementations are timely delivered and well tested. The release management practices must fit to the software development methodologies and should include the following:

- Maintenance of code versions in the local Git repositories of the tool developers
- Plan for frequent external releases that should be placed to the project Git repository
- From internal to external code version releases, the tool developers should perform unit tests to verify that the envisaged functionalities of the tool are properly propagated
- In the context of WP:D-7, the released code versions should be assessed on whether persistency and storage are affected. Then, the appropriate integration tests must be performed.
- If released code versions successfully pass the integration tests, they are fed to the use case instantiation pipeline in WP:D-7. Otherwise, the responsible tool developer is notified of the identified software deficiencies.

The detailed release management process will be updated and finalised in WP:D-7.

The precise and well-described documentation of the A4Cloud tools is also a key success for smooth, timely and efficient integration. In that respect, A4Cloud architecture guidelines should deliver a template structure for documenting the software produced in the course of the project. Such documentation will work as the reference material for the use case instantiation in WP:D-7, as well as a pillar towards attracting interested stakeholders for the target project communities.

The template structure for software documentation is proposed in the following:

- Overall high level design: this section should clarify the accountability requirements for delivering the functional specifications of the tool and the detailed internal development view of the architectural design. The latter should be placed in the context of the A4Cloud Reference Architecture and the operational perspective of accountability (the A4Cloud process view)
- Detailed design: this section should elaborate on the implementation details of the different modules and software components comprising the A4Cloud tools. Any existing background knowledge should be clearly documented in this section. For each module/software component, the input and output parameters should be defined.
- Tool Specification: this section should highlight the implementation details of the A4Cloud tool, including the used technologies and the detailed API specification of the tool.
- Prerequisites and installation: this section should mention the dependencies on the tools to external software, as well as a step-by-step guidance on the appropriate tool installation
- Integration scenarios: this section should provide examples of use in the context of the A4Cloud operational processes to support the accountability practises. Any interaction to other A4Cloud tools should be highlighted here.

4.12 Proposed standards and technologies

The following Table 1 summarises the proposed technologies and the standards to be used during the secure software development activities of the A4cloud project. This is a provisional list, which is subject to future changes, based on the progress of the work and the requirements arising within the development period.

Table 1: Summary of proposed standards and technologies to be used in A4Cloud

Category	Proposed Standard / Technology
Data Access and Modelling Technologies	JSON / protocol buffer
Infrastructure Portability Technologies	OpenStack
Application Interoperability Frameworks	RESTful Web Services
Communication Frameworks and Technologies	Enterprise Service Bus (Mule ESB/Service Mix)
Software Development Programming Language	No restrictions, but Java is recommended
Identity Management	If needed: OAuth and OpenID
Policy Management	Accountable PrimeLife Policy Language (APPL)
Secure Communication	If needed: Transport Layer Security (TLS), Certificate Authority and a Public Key Infrastructure
Monitoring and Incident Response	*To be examined in WP:C-8*
UI development	Web-based development, agnostic to programming language, but based on WP:C-7 analysis and WP:D-5 work
Application Server Technologies	Apache Tomcat / JBoss / JADE
Data processing	Hadoop (including HDFS and MapReduce) / RHIPE / R2Time

Category	Proposed Standard / Technology
Persistency and Data Storage	HBase / OpenTSDB
Code versioning	Git

5 Quality Assurance Assessment

Supplementing the secure software development lifecycle, the Software Qualification Testing Process is necessary to assess the maturity of the technical implementation and the alignment to the user requirements from a technical perspective. The technical assessment is supported by monitoring the technical parameters of the software performance and aims to determine how far the software integrated prototype meets the technical requirements and the functional specifications. It is an internal self-assessment made by technical experts engaged in the project development phase. The experts are allocated a dedicated area of work in the form of small projects and, during the testing and technical assessment, they assess the impact of the projects they are involved in on the basis of their perception of success.

Software developments can be tested, according to the established standards on software assurance process. This process aims to assess the efficiency of the platform functionalities and provide evidence that the integrated prototype is fully functional and available for release through a software assurance process. In principle, software assurance can be realised by evaluating both the software itself (the product) and how it has been developed (the process). Both aspects are important when targeting to support scalable functionalities. However, in research products, the software assurance of the process itself is almost impossible. It is mostly due to the constant change in both the design and the requirements of the software. In a research project, these changes occur during the project cycle itself, as a result of the ongoing research in the other work packages (reflected as changes in different components/building blocks of the architecture), especially, when agile software development is adopted. Thus, the software assurance process means testing the outcome of the research prototype, considering this as an integrated software, consisting of the individual components, which can be reached via remote service implementation.

ISO/IEC 12207:2008 standard [1], and especially the clause 7.2.3 about software quality assurance process and the clause 7.2.5 about software validation process define technical assessment practices that could be potentially be adopted in a research work, like A4Cloud, to provide the appropriate level of confidence that research has seriously considered testing and technical assessment of the proposed solution. Software validation is the confirmation that the software specifications conform to user needs and intended uses through examination and provision of objective evidence, and that the particular requirements implemented through software can be consistently fulfilled. Since software is usually part of a larger hardware system, the software validation typically includes evidence that all software requirements have been implemented correctly and completely.

Software validation is realised through quality models. In the past, different quality models have been proposed, each of which addresses different quality attributes that allow evaluating the developed software. Some of the most well-known are:

- McCall's model of software quality [125] (GE Model, 1977), which incorporates 11 criteria encompassing product operation, product revision, and product transition.
- Boehm's spiral model [126] (1978) based on a wider range of characteristics, which incorporates 19 criteria. The criteria in both, this and the GE model, are not independent as they interact with each other and often cause conflicts.
- ISO 9126-1 [127] incorporates six quality goals, each goal having a large number of attributes. These six goals are then further split into sub-characteristics, which represent measurable attributes (custom defined for each software product).

Recently the BS ISO/IEC 25010:2011 standard [128] about system and software quality models has replaced ISO 9126-1. Applying any of the above models is not a straightforward process. There are no automated means for testing software against each of the characteristics defined by each model. For each model, the final attributes must be matched against measurable metrics and thresholds for evaluating the results must be set. It is then possible to measure the results of the tests performed (either quantitative or qualitative/observed).

The ISO/IEC 25010:2011 standard is the most widespread reference model and it includes the common software quality characteristics that are supported by the other models. This standard defines two quality models providing a consistent terminology for specifying, measuring and evaluating system and software product quality:

- Quality in use model, which is composed of five characteristics that relate to the outcome of interaction with the system and characterises the impact that the product can have on the stakeholders.
- Product quality model, which is composed of eight characteristics that relate to static properties of software and dynamic properties of the computer system.

In the product quality model, the eight characteristics can be further divided into sub-characteristics, are shown in Figure 8.

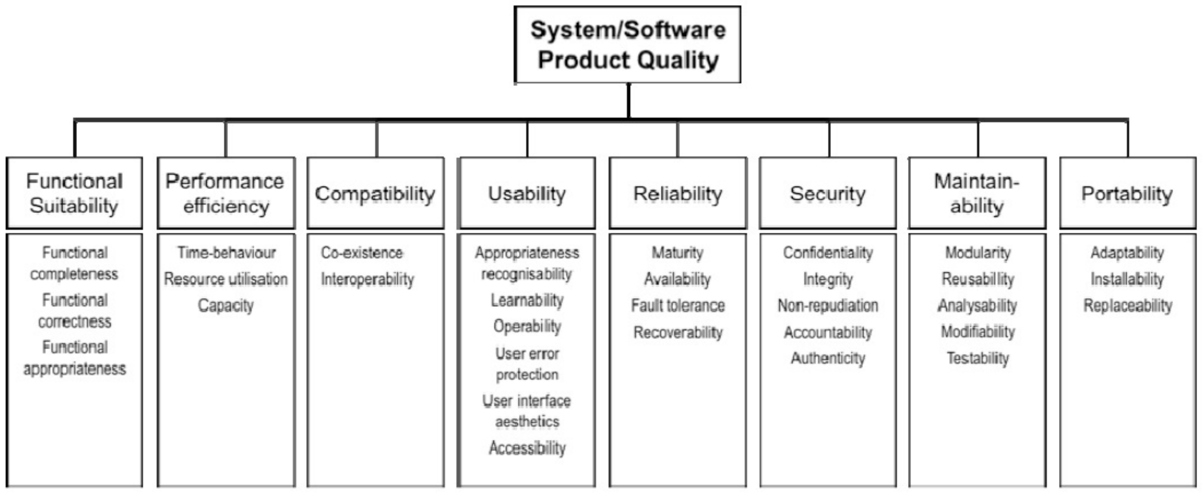

Figure 8: The ISO/IEC 25010:2011 system/software quality model characteristics [128]

For each of the sub-characteristics, a metric/measurable attribute is defined, along with thresholds. These metrics and thresholds are customised for each software solution. By evaluating the complete set of metrics, we are able to assess the overall quality of the software and the percent to which we were able to meet the user requirements (reflected to system specifications and functionalities) defined during the design phase of the project.

The software quality model of the BS ISO/IEC 25010:2011 standard is adapted on a case by case basis, in order to define appropriate metrics and to be able to evaluate the software capabilities. These metrics need to reflect the characteristics that they represent. They also need to allow appropriate measurements to be obtained, either through quantitative methods (e.g. by software tests/simulations, usability tests) or qualitative methods (e.g. through user observations). Three types of classes of metrics are defined in this standard:

- Internal metrics associated with static internal properties of a system such as number of function calls, number of rules.
- External metrics associated with dynamic external properties. These are metrics that are observable when the user interacts with the system (i.e. the user performs a task/function/operation and observes the response in the sense of time required, results obtained etc.).
- Quality-in-use metrics, which refer to metrics that evaluate the extent to which a system meets the needs of the user.

The quality assurance model involves software testing as well. The methodology for software testing is tightly coupled to the adopted software development methodology (as described in Section 4.4). In principle, two ways for testing a software product exist, namely the black box and the white box testing

[129], each of which exhibits certain advantages and drawbacks. Software testing can involve unit testing, system and integration testing and acceptance testing.

The implementation of the Software Qualification Testing Process through a standardised quality assurance framework, like ISO/IEC 25010:2011, including the testing framework, will be further considered in the next steps of the Reference Architecture and will be elaborated in detail as part of the work in T:D-2.3.

6 Conclusions and Next Steps

This document approached the architecture guidelines and principles for the design of the A4Cloud Reference Architecture. It presented the key directions that should drive this architecture and the different views and perspectives, through which the architecture should be realised and modelled. The document, also, elaborated on the way that the Reference Architecture will be instantiated and through which standards, technologies, tools and frameworks it will be developed to match the needs of real business cases.

The presentation of the design aspects of the A4Cloud Reference Architecture were based on the ISO/IEC 12207-2008 software implementation processes. For each process, specific references on how they will be realised in the forthcoming work of WP:D-2 were provided. In addition to it, a connection between the design and development of the A4cloud Reference Architecture with the detailed list of candidate technological standards, tools and frameworks to support the implementation and integration activities was drawn.

The guidelines and principles defined in this document will be used in the remaining work of WP:D-2 to drive the specification of the A4Cloud Reference Architecture. The progress of the design work, along with the actual implementation of the Reference Architecture in the context of the selected business scenario and accountability oriented use cases will assess the appropriateness of these guidelines, as well as it will reveal whether the assumptions made were suitable and sufficient to support the applicability of the Architecture to multiple and of wide scope business cases. Early assessment that will be made through the work in WP:D-6 and WP:D-7 mainly will potential drive calibrations, improvements and enhancements to the Reference Architecture, while they will leverage its completeness in the end of this iterative approach.

As shown in this deliverable, the basis for the A4Cloud Reference Architecture is a set of existing efforts in different SDOs, which try to approach the problem of data protection in cloud computing. Although the purpose of this architecture is to be sustainable and long living, it is expected that the exercise for instantiating it to real life cases will lead to (even cascading) revisions. But the ultimate scope is to discover whether new standards and practices can advance the current status of the cutting-edge technologies used in modern distributed information systems and leverage the notion of being accountable when handling personal and corporate data in the cloud.

7 References

[1] ISO/IEC 12207, “Systems and software engineering - Software life cycle processes”, IEEE Std. 12207-2008, Second edition, 2008-02-01.

[2] Ian Sommerville, “Software Engineering – Ninth Edition”, Addison-Wesley, 2011, ISBN-13: 978-0-13-703515-1 and ISBN-10: 0-13-703515-2.

[3] OASIS, Reference Model for Service Oriented Architecture 1.0, August 2006

[4] CADEC 2004, Reference Architecture - a foundation for successful projects, http://callistaenterprise.se/download/18.26813c1d126d398877680004948/Reference_Architecture.pdf, 2004

[5] VMWARE, PCI DSS 2.0 – “Validated Reference Architecture”, June 2013

[6] UK Government, “UK Government Reference Architecture (UKRA) - Version 1.0”, Government ICT Strategy, March 2012

[7] Office of the Assistant Secretary of Defense Networks and Information Integration (OASD/NII), “Reference Architecture Description, June 2010.

[8] Gerrit Muller and Eirik Hole, “Reference Architectures; Why, What and How”, Embedded Systems Institute and Stevens Institute of Technology, White Paper Resulting from Architecture Forum Meeting, March 12 & 13, 2007 (Hoboken NJ, USA), publication made on June, 2007.

[9] NIST, “Cloud Computing Reference Architecture”, Special Publication 500-292, September 2011

[10] The Open Group Architecture Framework (TOFAG): http://pubs.opengroup.org/architecture/togaf8-doc/arch/toc.html, last accessed 2013/11/20.

[11] Robert Cloutier, Gerrit Muller, Dinesh Verma, Roshanak Nilchiani, Eirik Hole, Mary Bone, “The Concept of Reference Architectures”, Systems Engineering, Volume 13, Issue 1, pages 14–27, Spring 2010, DOI: 10.1002/sys.20129, 2009 Wiley Periodicals, Inc.

[12] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description

[13] Antonio Vallecillo, RM-ODP: The ISO Reference Model for Open Distributed Processing

[14] ITU, “X.805 : Security architecture for systems providing end-to-end communications”, October, 2003

[15] DMTF, “Architecture for Managing Clouds”, DSP-IS0102, 2010-06-18

[16] https://cloudsecurityalliance.org/research/eawg/, last accessed 2013-11-20

[17] Cloud Computing Use Case Discussion Group, “Cloud Computing Use Cases”, white paper, Version 4.0, 2 July 2010, http://cloudusecases.org/.

[18] Distributed Management Task Force (DMTF) - Open Cloud Standards Incubator, “Use Cases and Interactions for Managing Clouds”, white paper, version: 1.0.0, publication date: 2010-06-18, document number: DSP-IS0103.

[19] ITU-T, Focus Group on Cloud Computing: “Part 2: Functional requirements and reference architecture”, Technical Report, version 1.0, February 2012.

[20] Microsoft, Security Development Lifecycle SDL Process Guidance Version 5.2, May 23, 2012

[21] N. Davis, Secure Software Development Life Cycle Processes, Carnegie Mellon University, Last revised: July 31, 2013

[22] J. Madison, Agile- Architecture Interactions, IEEE Software, March/April 2010

[23] Software development methodology”, http://en.wikipedia.org/wiki/Software_development_methodology, last accessed 2013/09/20.

[24] IT Knowledge Portal, http://www.itinfo.am/eng/software-development-methodologies/, last accessed 2013/11/20.

[25] Jurgen Appelo, The Definitive List of Software Development Methodologies: http://www.noop.nl/2008/07/the-definitive-list-of-software-development-methodologies.html#sthash.5hfBDNBl.dpuf, July 2008

[26] Successful Software development methods, http://www.program-ace.com/articles/development-practice/development-methodology/

[27] Pelican Engineering, “The Software Development Life Cycle (SDLC)”, Document ID: REF-0-02, Version: 2.0

[28] Rational Unified Process, “Best Practices for Software Development Teams, Rational Software White Paper, TP026B, Rev 11/01

[29] Robert C. Martin, Agile Software Development: Principles, Patterns and Practices, October 25, 2002, ISBN-10: 0135974445, ISBN-13: 978-0135974445

[30] Principles behind the Agile Manifesto: http://agilemanifesto.org/principles.html, last accessed 2013/09/20

[31] http://agilemanifesto.org/principles.html, last accessed 2013/09/20.

[32] Lee Badger, Robert Bohn, Ramaswamy Chandramouli, Tim Grance, Tom Karygiannis, Robert Patt-Corner, Jeffrey Voas, “Cloud Computing Use Cases”, Acknowledgments: The May 11, 2010 Gaithersburg Use Case workshop participants, Babak Johromi, Hemma Prafullchandra, and Gregg Brown, http://www.nist.gov/itl/cloud/use-cases.cfm

[33] Object Management Group - Business Process Modelling Notation (BPMN), www.bpmn.orghttps://netbeans.org/, last accessed 2013/09/20.

[34] Object Management Group - Unified Modelling Language (UML): www.uml.org, last accessed 2013/09/20.

[35] http://www.fmc-modeling.org/fmc-and-tam, last accessed 2013/12/03.

[36] SAP, Standardised Technical Architecture Modelling: Conceptual and Design Level, Version 1.0, March 2007

[37] Cristina Venera GEAMBASU, 2012, "BPMN vs. UML Activity Diagram for Business Process Modeling," Journal of Accounting and Management Information Systems, Faculty of Accounting and Management Information Systems, The Bucharest University of Economic Studies, vol. 11(4), pages 637-651, December 2012

[38] Peter Tabeling, Bernhard Gröne, Integrative Architecture Elicitation for Large Computer Based Systems, IEEE Conference "Engineering of Computer Based Systems", Washington D.C, 2005

[39] The Open Group - Cloud Computing Portability and Interoperability, http://www.opengroup.org/cloud/cloud_iop/cloud_port.htm, last accessed 2013/09/20.

[40] CDMI, http://www.snia.org/tech_activities/standards/curr_standards/cdmi

[41] XML, http://www.w3.org/XML/, last accessed 2013/09/20.

[42] JSON, http://www.json.org/, last accessed 2013/09/20.

[43] Protocol Buffers, https://developers.google.com/protocol-buffers/, last accessed 2013/09/20.

[44] Resource Description Framework (RDF), http://www.w3.org/RDF/, last accessed 2013/09/20.

[45] OWL Web Ontology Language, http://www.w3.org/TR/owl-features/, last accessed 2013/09/20.

[46] OpenStack Cloud Computing Software http://www.openstack.org/software/, last accessed 2013/09/20.

[47] OpenNebula.org, http://opennebula.org/about:abouthttp://www.openstack.org/software/, last accessed 2013/09/20.

[48] http://www.saphana.com/community/about-hana/cloud-platform, last accessed 2013/11/20

[49] http://www.saphana.com/community/about-hana, last accessed 2013/11/20

[50] CY13-Q2 Community Analysis — OpenStack vs OpenNebula vs Eucalyptus vs CloudStack, http://www.eucalyptus.com/blog/2013/07/03/cy13-q2-community-analysis-%E2%80%94-openstack-vs-opennebula-vs-eucalyptus-vs-cloudstackhttp://www.openstack.org/software/, last accessed 2013/09/20.

[51] Top 100 Cloud Services Providers (CSPs) List And Research: http://talkincloud.com/tc100

[52] Top 10 cloud computing providers 2012: http://searchcloudcomputing.techtarget.com/photostory/2240149038/Top-10-cloud-providers-of-2012/1/Introduction

[53] Citrix Vendor Landscape: Cloud Management Solutions: http://www.citrix.com/content/dam/citrix/en_us/documents/products/info-tech_research_group_cloud_management_vendor_landscape.pdf, last accessed 2013/11/20

[54] http://www.vmware.com/files/pdf/cloud/VMware-Taneja-Group-An-Overview-Of-The-Cloud-Market.pdf, last accessed 2013/11/20

[55] Info-tech, Vendor Landscape: Cloud Management Solutions, October 2012

[56] Leonard Richardson, Sam Ruby, “RESTful Web Services: Web services for the real world”, O'Reilly Media, May 2007, ISBN: 978-0-596-52926-0, ISBN 10: 0-596-52926-0

[57] James McGovern, Oliver Sims, Ashish Jain, Mark Little (Author), Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations, Springer; April 28, 2006, ISBN-10: 140203704X

[58] http://www.osgi.org/Main/HomePage, last accessed 2013/09/20.

[59] David Chappell, "Enterprise Service Bus", O‘Reilly: June 2004, ISBN 0-596-00675-6

[60] GAA Inc., http://www.gaaconsulting.com/soa/, last accessed 2013-11-20

[61] http://www.mulesoft.org/, last accesses 2013-11-20

[62] http://servicemix.apache.org/home.html, last accesses 2013-11-20

[63] http://www.eclipse.org/swordfish/, last accesses 2013-11-20

[64] Wikipedia, “A comparison of Integrated Development Environments”, http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments, last accessed 2013/09/20.

[65] Eclipse Foundation, www.eclipse.org, last accessed 2013/09/20.

[66] The Plug-in Development Environment (PDE), http://www.eclipse.org/pde/, last accessed 2013/09/20.

[67] Equinox, http://www.eclipse.org/equinox/, last accessed 2013/09/20.

[68] OSGi Alliance, http://www.osgi.org/ , last accessed 2013/09/20.

[69] Eclipse Plugin Development, http://eclipsepluginsite.com/, last accessed 2013/09/20.

[70] NetBeans IDE, https://netbeans.org/, last accessed 2013/09/20.

[71] The Google Go Project: http://golang.org/project/, last accessed 2013/09/20.

[72] OpenID, http://openid.net/, last accessed 2013/09/20.

[73] Open authorisation: http://oauth.net, last accessed 2013/09/20.

[74] OASIS Standard. eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013.

[75] Claudio A. Ardagna, Laurent Bussard, Sabrina De Capitani Di Vimercati, Gre- gory Neven, Stefano Paraboschi, Eros Pedrini, Stefan Preiss, Dave Raggett, Pierangela Samarati, Slim Trabelsi, and Mario Verdicchio. Primelife policy language. http://www.w3.org/2009/policy-ws/papers/Trabelisi.pdf, 2009.

[76] Alistair Barros and Daniel Oberle. Handbook of Service Description: USDL and Its Methods. Springer Publishing Company, Incorporated, 2012

[77] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. SLAng: A Lan- guage for Defining Service Level Agreements. In Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems, FTDCS ’03, pages 100–, Washington, DC, USA, 2003. IEEE Computer Society.

[78] W3C. Platform for Privacy Preferences. http://www.w3.org/P3P

[79] Moritz Y Becker, Alexander Malkis, and Laurent Bussard. S4p: A generic language for specifying privacy preferences and policies. Microsoft Research, 2010.

[80] A4Cloud Deliverable C-4.1: Policy Representation Framework. (in progress)

[81] IETF Transport Layer Security Protocol (TLS): tools.ietf.org/html/rfc5246, last accessed 2013/11/20.

[82] R.B. Bohn, J. Messina, Fang Liu, Jin Tong, and Jian Mao, "NIST Cloud Computing Reference Architecture," 2011, pp. 594-596.

[83] http://www.nagios.org/, last accessed 2013/09/20.

[84] http://ganglia.sourceforge.net/, last accessed 2013/09/20.

[85] http://aws.amazon.com/cloudwatch/, last accessed 2013/09/20.

[86] http://www.rackspace.com/cloud/monitoring/, last accessed 2013/09/20.

[87] David Abramowski, “Monitoring Applications in the Cloud”, http://abramowski.sys-con.com/node/870088,

[88] George Spanoudakis, Christos Kloukinas, and Khaled Mahbub, "The SERENITY Runtime Monitoring Framework," in Security and Dependability for Ambient Intelligence, Spyros Kokolakis, Antonio Mana Gomez, and George Spanoudakis, Eds.: Springer US, 2009, ch. 13, pp. 213-237.

[89] Doelitzscher Frank, Reich Christoph, Knahl Martin, Passfall Alexander, and Clarke Nathan, "An agent based business aware incident detection system for cloud environments," Journal of Cloud Computing: Advances, Systems and Applications, 2012.

[90] Doelitzscher F, Reich C, Knahl MH, and Clarke NL, "Incident detection for cloud environments," in Third International Conference on Emerging Network Intelligence.

[91] http://mmonit.com/monit/, last accessed 2013/11/20

[92] G. Spanoudakis, E. Damiani, and A. Mana, "Certifying Services in Cloud: The Case for a Hybrid, Incremental and Multi-layer Approach," in High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium, 2012, pp. 175-176.

[93] Siani Pearson et al., "Accountability for Cloud and Other Future Internet Services," in 4th International Conference on Cloud Computing Technology and Science.

[94] Grobauer B. and Schreck T., "Towards incident handling in the cloud: challenges and approaches," in ACM workshop on Cloud Computing Security Workshop (CCSW), 2010.

[95] Savola R. and Heinonen P., "Security-measurability-enhancing mechanisms for a distributed adaptive security monitoring system," in 4th International Conference on Emerging Security Information, Systems and Technologies, 2010.

[96] R.M. Savola, Habtamu Abie, J. Bigham, and D. Rotondi, "Innovations and advances in adaptive secure message oriented middleware," in Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference, 2010, pp. 288,289.

[97] Incident Handling in the Healthcare Cloud: Liquid Data and the Need for Adaptive Patient Consent Management: https://www.sans.org/reading-room/whitepapers/incident/incident-handling-healthcare-cloud-liquid-data-adaptive-patient-consent-ma-34007, last accessed 2013/11/20

[98] Vateva T., Adaptive Authentication for the Cloud, 2012, MSc. Thesis, TU Darmstadt.

[99] Soumitra (Ronnie) Sarkar, Mahindru Ruchi, Hosn Rafah A., Vogl Norbert, and Ramasamy HariGovind V., "Automated Incident Management for a Platform-as-a-Service Cloud," in USENIX Workshop on Hot Topics In Management of Internet, Cloud and Enterprise Networks and Services, 2011.

[100]Oracle GlassFish Server, http://www.oracle.com/us/products/middleware/application-server/oracle-glassfish-server/index.html, last accessed 2013/09/20.

[101]JBoss Community: http://www.jboss.org/overview/, last accessed 2013/09/20.

[102]http://felix.apache.org/site/index.html, last accessed 2013/09/20.

[103]http://tomcat.apache.org/, last accessed 2013/09/20.

[104]Java Agent DEvelopment Framework (JADE); http://jade.tilab.com/

[105]http://junit.sourceforge.net/javadoc/, last accessed 2013/09/20.

[106]http://hudson-ci.org/, last accessed 2013/09/20.

[107]http://jenkins-ci.org/, last accessed 2013/09/20.

[108]http://maven.apache.org/, last accessed 2013/09/20.

[109]http://ant.apache.org/, last accessed 2013/09/20.

[110]http://www.jfrog.com/home/v_artifactory_opensource_overview, last accessed 2013/09/20.

[111]http://trac.edgewall.org/, last accessed 2013/09/20.

[112]http://www.redmine.org/, last accessed 2013/09/20.

[113]http://www.sonarsource.org/, last accessed 2013/09/20.

[114]http://hadoop.apache.org/, last accessed 2013/11/29.

[115]Hadoop Beginners Guide, Garry Turkington, Packt Publishing

[116]http://www.datadr.org/ last accessed 2013/11/29.

[117]Agrawal, Bikash, "Analysis of large time-series data in OpenTSDB", 2013, Publisher: University of Stavanger, Norway, Series/Report no.: Masteroppgave/UIS-TN-IDE/2013

[118]http://www.mysql.com/, last accessed 2013/09/20.

[119]http://www.postgresql.org/, last accessed 2013/09/20.

[120]http://www.mongodb.org/, last accessed 2013/09/20.

[121]http://hbase.apache.org/, last accessed 2013/11/29.

[122]Open Time Series Database (OpenTSDB), http://opentsdb.net/, last accessed 2013/11/29.

[123]http://subversion.apache.org/, last accessed 2013/09/20.

[124]https://git.wiki.kernel.org/index.php/GitSvnComparison, last accessed 2013/09/20.

[125]J.A. McCall, P.K. Richards, and G.F. Walters, "Factors in Software Quality," vols. 1, 2, and 3, AD/A-049-014/015/055, Nat'l Tech. Information Service, Springfield, Va., 1977.

[126]Barry W. Boehm, " A spiral model of software development and enhancement", TRW, Defence Systems Group

[127]ISO/IEC 9126-1:2001, Software engineering -- Product quality -- Part 1: Quality model

[128]BS ISO/IEC 25010:2011, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models

[129]I. Jovanovic , Software Testing Methods and Techniques, Transactions on Internet research, January 2009

8 Glossary

Glossary terms adopted from WP:C-2.

Term	Description
A4Cloud	Accountability for Cloud and Other Future Internet Services
Access Control	The process of granting or denying specific requests: 1) for obtaining and using information and related information processing services; and 2) to enter specific physical facilities (e.g., Federal buildings, military establishments, and border crossing entrances).
Access Control Policy	The set of rules that define the conditions under which an access may take place.

Term	Description
Accountability Attributes	Conceptual elements of accountability as used across different domains.
Accountability Mechanisms	Diverse processes, non-technical mechanisms and tools that support accountability practices.
Accountability Practices	Emergent behaviour characterising accountable organisations.
Accountability-based Approach	An accountability-based approach to data governance is characterised by its focus on setting privacy-protection goals for organisations based on criteria established in current public policy and on allowing organisations discretion in determining appropriate measures to reach those goals.
Accountable Organization	An accountable organisation demonstrates commitment to accountability, implements data privacy policies linked to recognised outside criteria, and establishes performance mechanisms to ensure responsible decision-making about the management of data consistent with organisation policies.
Cloud Computing	Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
Cloud Ecosystem	A cloud computing business ecosystem (cloud ecosystem) is a business ecosystem of interacting organizations and individuals - the actors of the cloud ecosystem - providing and consuming cloud services.
Information Security	The protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability.
Hybrid Cloud	The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).
Personal Data	'Personal data' shall mean any information relating to an identified or identifiable natural person ('data subject'); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity.
Policy	A set of rules related to a particular purpose. A rule can be expressed as an obligation, an authorization, a permission, or a prohibition. Not every policy is a constraint. Some policies represent an empowerment.

Term	Description
Policy Enforcement	The execution of a policy decision.
Private Cloud	The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units).
Public Cloud	The cloud infrastructure is provisioned for open use by the general public.
Reference Architecture	A Reference Architecture is, in essence, a predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed and proven for use in particular business and technical contexts, together with supporting artifacts to enable their use
Resource Pooling	The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
Violation	A behavior contrary to that required by a rule.
Vulnerability	Weakness of an asset or control that can be exploited by a threat. Any circumstance or event with the potential to adversely impact an asset through unauthorized access, destruction, disclosure, modification of data, and/or denial of service.

9 Index of Figures

10 Index of Tables