Aston University

Some pages of this thesis may have been removed for copyright restrictions.

If you have discovered material in Aston Research Explorer which is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please read our Takedown policy and contact the service immediately (openaccess@aston.ac.uk)

KNOWLEDGE-CENTRIC AUTONOMIC SYSTEMS

ALINA PATELLI

Doctor of Philosophy

ASTON UNIVERSITY

April 2016

Abstract

Autonomic computing revolutionised the commonplace understanding of proactiveness in the digital world by introducing *self-managing* systems. Built on top of IBM's structural and functional recommendations for implementing intelligent control, autonomic systems are meant to pursue high level goals, while adequately responding to changes in the environment, with a minimum amount of human intervention. One of the lead challenges related to implementing this type of behaviour in practical situations stems from the way autonomic systems manage their inner representation of the world. Specifically, all the components involved in the control loop have shared access to the system's knowledge, which, for a seamless cooperation, needs to be kept consistent at all times.

A possible solution lies with another popular technology of the 21st century, the *Semantic Web*, and the knowledge representation media it fosters, ontologies. These formal yet flexible descriptions of the problem domain are equipped with reasoners, inference tools that, among other functions, check knowledge consistency. The immediate application of reasoners in an autonomic context is to ensure that all components share and operate on a logically correct and coherent "view" of the world. At the same time, ontology change management is a difficult task to complete with semantic technologies alone, especially if little to no human supervision is available. This invites the idea of delegating change management to an autonomic manager, as the intelligent control loop it implements is engineered specifically for that purpose.

Despite the inherent compatibility between autonomic computing and semantic technologies, their integration is non-trivial and insufficiently investigated in the literature. This gap represents the main motivation for this thesis. Moreover, existing attempts at provisioning autonomic architectures with semantic engines represent bespoke solutions for specific problems (load balancing in autonomic networking, deconflicting high level policies, informing the process of correlating diverse enterprise data are just a few examples). The main drawback of these efforts is that they only provide limited scope for reuse and cross-domain analysis (design guidelines, useful architectural models that would scale well across different applications and modular components that could be integrated in other systems seem to be poorly represented).

This work proposes KAS (Knowledge-centric Autonomic System), a hybrid architecture combining semantic tools such as:

- an ontology to capture domain knowledge,

- a reasoner to maintain domain knowledge consistent as well as infer new knowledge,

- a semantic querying engine,

- a tool for semantic annotation analysis

with a customised autonomic control loop featuring:

- a novel algorithm for extracting knowledge authored by the domain expert,

- "software sensors" to monitor user requests and environment changes,

- a new algorithm for analysing the monitored changes, matching them against known patterns and producing plans for taking the necessary actions,

- "software effectors" to implement the planned changes and modify the ontology accordingly.

The purpose of KAS is to act as a blueprint for the implementation of autonomic systems harvesting semantic power to improve self-management. To this end, two KAS instances were built and deployed in two different problem domains, namely self-adaptive document rendering and autonomic decision

support for career management. The former case study is intended as a desktop application, whereas the latter is a large scale, web-based system built to capture and manage knowledge sourced by an entire (relevant) community. The two problems are representative for their own application classes – namely desktop tools required to respond in realtime and, respectively, online decision support platforms expected to process large volumes of data undergoing continuous transformation – therefore, they were selected to demonstrate the cross-domain applicability (that state of the art approaches tend to lack) of the proposed architecture. Moreover, analysing KAS behaviour in these two applications enabled the distillation of design guidelines and of lessons learnt from practical implementation experience while building on and adapting state of the art tools and methodologies from both fields.

KAS is described and analysed from design through to implementation. The design is evaluated using ATAM (Architecture Tradeoff Analysis Method) whereas the performance of the two practical realisations is measured both globally as well as deconstructed in an attempt to isolate the impact of each autonomic and semantic component. This last type of evaluation employs state of the art metrics for each of the two domains. The experimental findings show that both instances of the proposed hybrid architecture successfully meet the prescribed high-level goals and that the semantic components have a positive influence on the system's autonomic behaviour.

Acknowledgements

I dedicate this work to my mother, Carmen Patelli. When I am blindsided by what I think I want, she points out what I actually need.

The road so far would have been a lot harder without my father and engineering role model, Dinu Patelli.

My deepest gratitude to my supervisor, Dr Hai Wang, my research collaborators (and friends), Dr Aniko Ekart, Dr Peter Lewis, Prof Ian Nabney, Dr Radu Calinescu and, last but not least, my PhD colleagues, Alex, Yanna, Sandra, Ana, Ali, Nick and Aman.

Contents

List of Abbreviations

ABLE	Agent Building and Learning Environment
ACME	Architectural Description of Component-based Systems
ACT	Autonomic Computing Toolkit
ADL	Architectural Description Language
ANS	Autonomic Nervous Systems
ATAM	Architecture Trade-off Analysis Method
CDS	Career Decsion Support
CSS	Cascading Style Sheets
CWA	Closed World Assumption
DAML	DARPA Agent Modelling Language
DARPA	Defence Advances Research Projects Agency
DL	Description Logic
ECA	Event Condition Action
GATE	General Architecture for Text Engineering
GCG	Good Careers Guide
GPAC	General Purpose Autonomic Computing
HTML	Hyper Text Markup Language
IRI	Internalised Resource Identifier
JACS	Joint Academic Coding System
KAoS	Knowledgeable Agent-oriented System
KAS	Knowledge-centric Autonomic System
KPAT	KAoS Policy Administration Tool
KT	Knowledge Translator
MAPE-K	Monitor Analyse Plan Execute - Knowledge
NN	Neural Network
OE	Ontology Engineering
OL	Ontology Learning
OOP	Object Oriented Programming
OWA	Open World Assumption
OWL	Web Ontology Language
OWL-POLAR	OWL-based Policy Language for Agent Reasoning
PDDL	Planning Domain Definition Language
ProProST	Probabilistic Property Specification Templates
RDF	Resource Description Framework
RDFS	RDF Schema
RIF	Rule Interchange Format
SAR	Self-Adaptive document Rendering
SLA	Service Level Agreement
SMART	Specific Measurable Achievable/Agreed/Actionable Realistic/Relevant Timed
SOC	Standard Occupational Classification
SPARQL	Simple Protocol and RDF Query Language
SWRL	Semantic Web Rule Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
xADL	highly-eXtensible ADL
XML	eXtensible Markup Language

List of Tables

List of Figures

Introduction

1.1 Problem Statement

The beginning of the third millennium brings forward two remarkable challenges in the field of computing. The first is formulated by Tim Berners-Lee et al. in their May, 2001 article [23], where the authors observe the state of the web, analyse its limitations and suggest a foundational shift in its structure. More specifically, this seminal piece of work advocates the transformation of the *web of documents*, a heterogeneous collection of online resources chiefly meant for human consumption, into a *web of data*, a network of uniformly formatted knowledge that can be processed by machines. This is made possible by annotating every piece of information on the web with metadata, namely small units of text (appropriately called *tags*), expressed in some well structured language, meant to bring context to the resources they are attached to. By reading and interpreting these tags, machines are envisaged to break the syntactic barrier of processing information and become capable of understanding its *meaning*. This type of behaviour would bring about a new web, a *semantic web*. Later that same year, IBM publishes a manifesto [81] issuing a strong warning about the galloping increase of IT systems complexity and its backlash against the very heightened computing power it was meant to make possible. The proposed solution is *autonomic computing*, a biologically inspired model that promotes *self-managing* IT systems. Much like the human autonomic nervous system, self-managing IT structures are governed by a manager that takes care of low-end yet vital operations such as integration with other systems, download and installation of updates, protection against malware, etc. [101]. This would hide a significant portion of the IT system's complexity from the human administrator who is thus allowed to invest energy in high level tasks such as goal/policy definition and business requirements management.

The semantic web and autonomic computing initiatives are connected by more than their date of birth. There has been significant work on integrating semantic technologies into the Internet of Things (IoT) application domain [190, 186, 16], that also makes use of autonomic elements – to provide just one example of research that bridges the gap between the two fields. To further support this compatibility claim, let us outline some of the key strengths and open challenges of both fields. As mentioned before, the semantic web can be practically realised by annotating online resources with metadata (semantic tags). Although some commercial applications, such as Flickr[1] or Delicious[2], support manual tagging, given the size of the current web, an automated annotation solution would be preferable. To address that, the semantic research community suggested extracting annotations from ontologies

---

[1] https://www.flickr.com/

[2] https://delicious.com/

[23, 78, 73, 8, 54, 191]. Ontologies are formal models of a given domain, usually written in a language from the Resource Description Framework (RDF[3]) family, such as the Web Ontology Language (OWL[4]). Some well known examples of ontologies are Google Knowledge Graph [159], supporting the right hand side display of the search engine's results page, WordNet[5], an English language lexicon, DBpedia[6], organising structured Wikipedia information in a semantic format for better querying, OpenCyc[7], an ambitious project to build a general ontology about everything in the world, etc. (other smaller, domain-oriented ontologies can be found in dedicated libraries [49]). This wide acceptance is driven by the main appeal of semantic technologies, stated below.

> Ontologies are equipped with reasoners, semantic inference tools capable of automatically verifying the logical correctness of the knowledge base.

Thus, there is a way to formally guarantee that metadata is extracted from a logically consistent repository. However, the domains modelled by ontologies are dynamic, continuously changing by incorporating new information, modifying existing data and discarding obsolete records. In this context, an important challenge can be formulated, as follows.

> Ontologies need to be (automatically) maintained to keep up with the frequent transformations of the knowledge domain they are modelling.

A similar analysis can be carried out for autonomic systems. Autonomic behaviour (a four-faceted concept [101] comprising self-configuration, self-optimisation, self-healing and self-protection) is achieved by plugging an *autonomic manager* into a traditional (legacy) IT system, also called the *managed resource* (Fig. 1.1). The operation of the manager is based on a closed control loop with four stages: *monitor*, *analyse*, *plan* and *execute*, all of which entail some level of interaction with a central repository of *knowledge* about the system itself and its environment. This set-up is referred to as the MAPE-K loop and is widely used in both research (ranging from generic toolkits such as ABLE [24] and GPAC [32] to concrete solutions for dynamic resource management in distributed systems [1], smart home applications [172] and wireless sensor networks management [89]) and industry (IBM's autonomic computing toolkit [91], Fujitsu's organic computing architecture [98] and other successful implementations by Oracle, HP, Microsoft and Intel [103]). The lead advantage of the MAPE-K loop is that it allows any system based on it to manage change, that is, adapt to environment dynamics in order to continue achieving a set of goals prescribed in the *policy* document. In short, this is done by *monitoring* the environment via *sensors* to detect changes, *analysing* the acquired data by matching it against available models or previously detected patterns (usually, part of the system *knowledge*), *planning* a set of actions to respond to the changes and *executing* those actions at a practical level via *effectors*. The lead advantage (included below) of autonomic systems is a direct consequence of this process.

---

[3]http://www.w3.org/RDF/

[4]http://www.w3.org/2001/sw/wiki/OWL

[5]http://wordnet-rdf.princeton.edu/

[6]http://wiki.dbpedia.org/

[7]http://www.opencyc.org/

> Autonomic systems intrinsically maintain their knowledge base up-to-date with respect to changes in the environment.

Self-managing computing is performed in realtime, therefore, the MAPE components will access and modify the knowledge base concurrently. This gives rise to the following challenge.

> Autonomic systems require a means to ensure that their knowledge repository is consistent (logically correct) at all times.
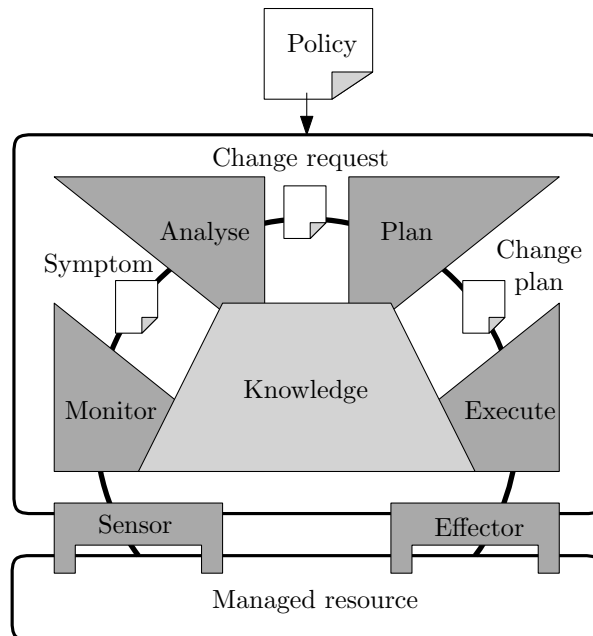
Fig. 1.1: The simplified architecture of a traditional autonomic computing element - adapted from [90]

By analysing the previously identified strengths and challenges, it is fairly straightforward to notice their overlap. One of the main open issues in semantic web research is the effective maintenance of ontologies as the domains they model are continuously changing. Autonomic systems are designed to accomplish just that: manage change whilst still optimally pursuing their prescribed goals. They are however hindered by the likelihood of their knowledge base becoming inconsistent due to concurrent access. Ontology reasoners are specifically built to prevent such a scenario by automatically verifying the logical correctness of the knowledge repository. This alignment between the semantic and autonomic fields invites further research into ways of combining the two technologies and thus improving them by enabling the exploitation of features from the complementary domain.

1.2 Research Questions

Knowledge is at the core of the autonomic control loop [90, 101], representing the manager's model of its environment and of its inner state. As shown in Fig. 1.1, all MAPE components use the central repository of information, *concurrently*, to inform their operation. Thus, the knowledge model has to be kept in a logically consistent state as well as up to date with the changes in the manager's environment and/or state. It follows naturally that the way knowledge is represented plays a key role in meeting those two requirements, however, there is limited focus in the literature on the systematic description of environment and inner state information. The existing approaches mainly fall under three categories.

- Knowledge is not permanently stored, instead, it is collected by sensors, during live operation, used to inform the MAPE components in the current stage of the control loop and then discarded [95, 165, 182].

- Knowledge is kept in some formal representation that usually takes the form of an architectural design language (ADL) [88, 110, 152, 50]. Specifically, an ADL representation decomposes knowledge in *components* linked by *connectors*, a graph-like structure that lends itself well to live updates and computationally inexpensive integrity checks.

- Knowledge resides inside the manager and informs the MAPE loop, as can be deduced from the overall system's behaviour, but the details of knowledge storage and management are undisclosed [27, 108].

There are several problems stemming from the knowledge representation strategies above. Firstly, relying only on sensor data captured in the current cycle of operation implies that the manager cannot learn from past experience, thus it becomes challenging to improve the MAPE components based on patterns extracted from historical data. Secondly, using ADLs or other formal languages restricts access to the data layer, as the knowledge model becomes difficult to understand by non-experts. Finally, if the knowledge layer is used to underpin autonomic behaviour in some application-specific way that is too complicated/trivial to be described in the relevant paper, then the proposed solution is unlikely to be reused across different problem domains.

In contrast to these shortcomings, ontologies offer a more flexible and intuitive knowledge storage solution. In effect, semantic knowledge repositories inherently support learning [23, 158] - reasoners infer new information, in an unsupervised way, thus allowing the manager to base its decisions on a comprehensive pool of data. Moreover, ontologies are usually written in RDF, a light logical language based on ⟨`subject`, `predicate`, `object`⟩ triplets. Consequently, semantically stored knowledge lends itself well to being visually rendered as a graph [58, 130, 164, 82, 167, 185], making ontologies more intuitive to navigate by non-expert users. Ultimately, ontologies represent a versatile technology, used in a variety of problem domains (see 3.2), and enjoy a wide industry acceptance as standardised solutions for medium to large scale knowledge representation.

In light of these advantages, the research community has proposed hybrid architectures, fitting autonomic managers with semantic knowledge layers. The recurring theme of these contributions is highlighting the hybrid systems' capacity to learn, that is, infer new information automatically [39, 149, 9] and use it to inform the autonomic control loop. On their quest to increase the acuity of

learning, the authors sometimes compromise good software engineering practices and seldom provide substantive design recommendations and/or lesson learnt, making it difficult to replicate the proposed hybrids in different problem domains. The evaluation of these semantically enhanced autonomic architectures is also in-house.

Given the state of the art summarised above, the main research question of this work involves the possibility of constructing effective, responsibly designed autonomic architectures, by storing the manager's knowledge in an ontology. This can be further broken down into two sub-questions:

- how to *design* an effective hybrid (autonomic and semantic) framework?

- how to *implement* the hybrid framework across different problem domains?

The first sub-question is approached in three stages. Firstly, a general purpose hybrid architecture, namely, an autonomic manager implementing a MAPE-K loop informed by an ontology, is proposed in chapter 4. The underpinning components are structured and assembled to meet the requirements, as presented in the literature, for obtaining effective autonomic behaviour. Secondly, the tools necessary to implement the architectural components are comprehensively described and implemented (chapter 4.3), to ensure the practicality of the suggested design. Thirdly, the methodology to assemble the architectural components coherently is provided in chapter 4.4, for the benefit of customising the suggested hybrid to other problem domains.

The second sub-question is addressed by building practical instances of the proposed hybrid architecture and deploying those in two different application domains - one where adaptability to the environment and speedy decision making are key (chapter 5) and one where domain knowledge is abundant, dynamic and community-curated (chapter 6).

1.3   Proposed Solution and Application Domains

In response to the research question formulated in the previous section, this work proposes a novel, hybrid autonomic architecture, where the knowledge centrepiece in Fig. 1.1 is replaced by an ontology. Since the focus of this research is on ways to exploit and enhance ontology properties (such as hierarchical structure [136], reasoning and querying capabilities [19], etc.) in order to improve the runtime performance of the autonomic ensemble, the suggested hybrid architecture can be viewed as knowledge-centric and is thus titled KAS (Knowledge-centric Autonomic System). The KAS architecture was customised for and tested in two different application domains, the first one dealing with self-adaptive document rendering and the second with career related decision support.

1.3.1   Self-adaptive Document Rendering

In the first application domain (see chapter 5 for a detailed description), the legacy system is a traditional document in electronic format, meant to be presented in front of an audience. Specifically, what is being managed is not the content of the document (as that does not change throughout the presentation), but its appearance, namely the size of the fonts used and the brightness of the display. These features are captured within the ontology alongside information about the environment, such as the hour of the day and the audience's level of focus as picked up by a camera. The goal of KAS in this context is to

ensure that the presentation of the document is well received by maintaining the audience's focus at a satisfactory level. To exemplify the system's operation, let us assume that it is 4 pm and that the detected level of audience focus is medium. Since that level is likely to drop even more in the late hours of the afternoon, KAS will automatically increase font size as well as screen brightness to make the contents of the presentation easier to take in.

Media content adaptation was selected as an application domain for two reasons: it is a relevant topic in recent research [52, 122, 144, 192] and several related open issues can be addressed via a joint autonomic/semantic approach, as embodied by KAS. These issues are briefly presented below:

- The media content adaptation problem is formulated, chiefly, with respect to device constraints (e.g., maximum allowed resolution) and network limitations (e.g., available bitrate) [122, 144, 192]. The *consumption style* of the media user, namely personal requirements/preferences affecting the quality of experience, is not sufficiently considered [52]. In contrast, the proposed KAS implementation takes into account the viewer's level of concentration, as captured by sensors, and adjusts document rendering accordingly.

- In order to provide rendering devices with sufficient information for content adaptation, the A/V stream is accompanied by *manifest files* [52] - containing technical specifications, such as required codecs and native resolution - or by *metadata* [192] - expressing precomputed relationships between various sets of adaptation parameters. There is no mention about the language/format used for representing this additional information, nor about the decryption process. Overall, the authors of manifest files and/or metadata are most likely domain experts with a solid technical knowledge of the problem environment. To open the content adaptation domain to application developers who are not necessarily versed in the theory of A/V physics, KAS is structured in a modular fashion, where each architectural component is easily configurable and semi-automated.

- In [52], the authors argue that content adaptation techniques would benefit from a *unified standard*, in terms of rationalising their development, maintenance and evolution. Moreover, [192] advocates the *scalability* of content adaptation frameworks, which should be applicable/extensible to a wide range of media delivery/rendering system architectures. KAS answers these two challenges, as the traditional infrastructure presented in Fig. 1.1 is a *standards-based architectural building block* (as described in [90]). Several such building blocks may be coupled together to create a multi-layered autonomic computing architecture (illustrated in [90], Figure 2) that can allegedly scale up to any business model.

The core KAS architecture in Fig. 1.1 (where the knowledge block is an ontology) had to be tailored in order to fit the requirements of the media content adaptation problem. One modification is the addition of a discretisation mechanism (implemented by KAS's bespoke ontology learning algorithm) capable of classifying monitored data under several discrete categories, thus simplifying the definition of system states (the `State` hierarchy in Fig. 5.1). Another customisation involves the plan bank, a repository of known good quality plans, successful at driving the system to a state of increased utility (relative to the initial one). These plans are recycled when possible and improved when needed, demonstrating that KAS instances are capable of learning from their own experience. However, larger application domains may be difficult to model with a `State` hierarchy of reasonable size and, on the other hand, may require

a more detailed ontology representation of the managed resource entities and properties, a reality that the KAS framework is flexible enough to support. To demonstrate that the proposed KAS architecture does scale up to problems of a larger size (and of a different nature altogether), a second practical scenario is introduced.

1.3.2 Career-related Decision Support

The second application domain (analysed at length in chapter 6) consists in an online career recommendations platform (`https://gcg-test.codevate.com`[8]). Information about different careers (e.g., the names of different professions and the number of associated jobs, standard higher education codes assigned to career domains, relationships between careers, employer and job seeker profiles, etc.) is stored in an ontology and presented to the end user in the form of a graph. Graph nodes represent professions whereas edges illustrate the (parent to child or sibling to sibling) relationships between them. This sets the platform apart from traditional job search engines, where results are displayed in a list, thus obscuring the relationships between careers and ultimately failing to give the end user a global perspective on the career universe.

The main features available for the job seeker are tagging visited webpages with concepts from the ontology, querying the ontology for access to a graph segment relevant to a search keyword, editing the ontology by adding/deleting career concepts along with their relationships and viewing a personal ontology containing the network of career concepts used as tags in the user's browsing history. From an autonomic perspective, the career advice platform exhibits self-management (as a result of coupling it with an autonomic manager based on the KAS architecture) in that it *monitors* user requests such as ontology querying, editing or personal ontology generation, it *analyses* those requests in order to select the appropriate plan for action, it *plans* operations such as ontology classification (consistency verification) and segmentation (query running) and, finally, it *executes* the plan to display the appropriate ontology view for the end user.

Large scale knowledge graph visualisation (with a focus on careers) was selected as an application domain for the following reasons:

- Career related knowledge is voluminous (sourced by companies' websites, job search engines, statistics agencies[9], etc.) and heterogeneous (each of the previously enumerated providers publishes job information in a different format). Considering a central ontology imposes a consistent knowledge representation (RDF/OWL), thus facilitating further semantic processing such as reasoning (to automatically infer implicit information) and logical validation (difficult to achieve otherwise, as logical conflicts may arise across separate career knowledge repositories, even if they are formally correct individually).

- Another consequence of career knowledge being scattered across several platforms is the loss of connections. For example, a front end developer job advert on `indeed.co.uk` requires

---

[8]Domain knowledge provided by Ralph Lucas with the Good Careers Guide; web interface built by David Bennett with Codevate

[9]Statistical data about the UK higher education is available at `https://www.hesa.ac.uk/`, the American Institute of Physics publishes data about employment in their field at `http://www.aps.org/careers/statistics/`, general job statistics for the UK can be found at `http://ons.gov.uk/ons/taxonomy/index.html?nscl=Job+Statistics`, etc.

specific computing skills. A simple way for the potential candidate to link those requirements to courses she may have taken in university (and therefore determine if she is eligible for the job) is tracking down the JACS classification for subjects such as computer science (I100) or software design (I310)[10]. Even if the candidate is aware of both publishers (`indeed.co.uk` for the job requirements and the HESA website for JACS information), other useful data, such as the average number of jobs in front end development currently available, may require supplementary effort to gather. A centralised ontology attaches all related information to the relevant career graph node (users can access the JACS classification and associated volume of job offers by expanding the COMPUTER SCIENCE node in the graph). This approach not only reveals the way careers are related (e.g., computer science to software design and front end developing) but also provides quick access to statistical data associated to nodes, thus giving the end user a global, connected view of the career domain.

- The field of careers is constantly changing with professions being introduced or redefined to reflect technological trends, business requirements or shifting views of the wider community with respect to which careers are most needed in the current professional context. The ontology captures all moving parts of the career domain by allowing job seekers, employers and other stakeholders to edit the ontology in light of their latest developments. Another facet of change management is the interactive nature of the platform. Rather than displaying a static list of job results like most traditional providers, the proposed service is autonomic, therefore responsive to explicit cues (e.g., user requests) as well as implicit ones (changing organisation profiles that may trigger a modification in the compatibility score of job seekers registered on the system).

In order to fulfil the requirements of the large scale knowledge graph visualisation application domain, the ontology component in the KAS architecture (the knowledge block in Fig. 1.1) has to communicate with several other modules pertaining to the online functionality of the system (see Fig. 6.1). This allows the platform to scale to a much larger problem domain than that of self-adaptive document rendering and cater to the wider group of career recommendations seekers. Beyond demonstrating the versatility of the KAS architecture, this high impact, large scale application domain also enabled public domain deployment (outside the research community, thus benefiting from feedback from a larger audience), sounder performance testing (6.4) and the involvement of industrial partners[11] to keep development in line with realistic business needs.

1.4 Research Objectives and Contributions

The work briefly described in section 1.3 pursued the following SMART objectives in answer to the research questions proposed in 1.2:

O1 Develop a framework for a conceptual hybrid system combining autonomic principles (the MAPE-K loop) with semantic technologies (ontologies, semantic tagging and reasoning). The framework (and its problem-specific instances) will be evaluated both pre-implementation (using

---

[10]Codes retrieved from `https://www.hesa.ac.uk/content/view/102/143/1/8/`

[11]Good Careers Guide http://gcgchangeworks.com/, Codevate https://www.codevate.com/ and Capgemini https://www.uk.capgemini.com/

the Architecture Trade-off Analysis Method – ATAM[12] approach – detailed in 4.5) and post-implementation, by testing the performance of the resulting practical systems. This objective can be broken down as follows:

O1.1 Design the core KAS framework comprising an *architectural blueprint*, a *set of tools* to implement the various components of the proposed architecture and *a methodology* to describe the interactions between system modules and also act as a guide for application developers wishing to create bespoke KAS instances.

O1.2 Identify relevant application scenarios and create instances of the core framework to fit specific requirements of the chosen problems.

O1.3 Implement KAS instances by configuring and integrating all architectural components.

O2 Analyse the hybrid architecture by measuring offline features suggested in the literature as well as investigating the runtime performance of the implemented instances. The proposed analysis has three facets illustrated by the following sub-objectives:

O2.1 Evaluate KAS properties (such as design cohesion, learning capacity, domain knowledge coverage, etc. – see 5.4 and 6.4 for a complete list) employing metrics suggested in the literature. This will establish some common ground for comparing KAS against other autonomic and semantic platforms.

O2.2 Test the robustness of KAS implementations by running performance-oriented experiments and interpreting the numeric results.

O2.3 Reflect on the way autonomic and semantic principles influence the KAS architecture and implementation. Compile a set of lessons learnt from the development and experimentation process.

The pursuit of the objectives above has materialised in three types of contributions related to framework design, tool-supporting algorithms (along with their implementation) and analysis (offline, experimental and reflective). They are enumerated below:

D1 *design* of the KAS framework (architecture, tools and methodology);

D2 *design* of a KAS instance with a state-featuring ontology (Fig. 5.1);

D3 *design* of a KAS instance with a state-less ontology (Fig. 6.1);

I1 *implementation* for all autonomic and semantic components of the KAS instance with a state-featuring ontology;

I2 *implementation* for all autonomic and semantic components of the KAS instance with a state-less ontology;

E1 *experimental* performance analysis of the KAS instance with a state-featuring ontology deployed on the self-adaptive document rendering application;

---

[12]http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

E2 *experimental* performance analysis of the KAS instance with a state-less ontology deployed on the career-related decision support application;

R *reflection* on experimental results and overall development experience; synthesis of a set of lessons learnt during design and implementation (with focus on the impact of autonomic and semantic principles on platform building).

The mapping between the research objectives and the specific contributions of this work, along with the thesis chapter where the latter are discussed at length, is presented in Table 1.1.

Table 1.1: Contributions to objectives mapping

Contribution	Objective	Chapter
D1	O1.1	4
D2	O1.1, O1.2	5.2
D3	O1.1, O1.2	6.2
I1	O1.2, O1.3	5.1, 5.2
I2	O1.2, O1.3	6.1, 6.2
E1	O2.1, O2.2	5.4
E2	O2.1, O2.2	6.4
R	O2.3	7.2

Background

This chapter sets the core vocabulary and facilitates a basic understanding of *autonomic systems*, *semantic technologies* and *crossover platforms* incorporating principles and tools from both fields. The provided insight covers fundamental aspects such as structure, applications and open issues. Some general reflection is carried out to outline the need for and usage of hybrid architectures as well as the accompanying design challenges.

2.1 Autonomic Systems

The main purpose of autonomic computing is to partially shield the human operator from the complexity of IT systems functioning (and interacting) in the modern digital universe [101, 90]. The menial yet cumbersome tasks inherent to system administration can be made transparent to the human element in several ways: automatic system configuration based on operating conditions (a), proactive optimisation of resource use (b), unsupervised problem detection/diagnosis/correction (c) and intrinsic protection from threats (d). These can be easily assimilated to low level processes in the human body, for instance, contracting/dilating pupils in response to the intensity of light in the environment (a), increasing blood flow for better absorption of nutrients and oxygen (b), which also contributes to tissue regeneration and healing (c), and triggering the fight-or-flight reflex to respond to outside threats (d). If managed by the conscious mind, these vital yet minute operations would be our main preoccupation, hindering higher level, intellectual functions.

Fortunately, the human autonomic nervous system (ANS) is capable of effectively carrying out all previously mentioned tasks without conscious intervention. In computing, the equivalent of the ANS is the *autonomic manager*. When plugged into legacy systems (Fig. 1.1), it endows them with autonomic behaviour, giving rise to an *autonomic computing element*. An autonomic system (Fig. 2 in [90]) is formed of one or several autonomic elements in interaction and is characterised by one paramount quality: *self-management*. This is considered the essence of autonomic computing [101, 90] and essentially refers to IT systems' capacity of proactively pursuing high level goals with little or no assistance from human operators. [90] best captures the essence of self-management by referring to it as the process of "using technology to manage technology".

The autonomic manager facilitates four types of *self-management*, addressing each of the previously described facets of complexity: self-configuration (a), self-optimisation (b), self-healing (c) and self-protection (d). According to IBM [88], depending on the way the MAPE-K loop is implemented, the four autonomic properties may be attained to various degrees:

- **basic** - the computing system is merely a *monitor* of its environment and it is up to human operators to make configuration, optimisation, etc. decisions based on interpreting monitoring logs;

- **managed** - the computing system runs *smart monitoring* techniques by filtering out noise, organising collected data based on its source and analysing it to highlight patterns, thus simplifying the work of human operators;

- **predictive** - smart monitoring is followed by a *comprehensive analysis* of collected data resulting in a set of suggested actions for human operators to approve and execute;

- **adaptive** - the computing system proactively carries out the previously compiled set of actions in pursuit of *high level goals*, mostly in the form of service level agreements (SLAs), stating the quality of service agreed upon by both providers and customers;

- **autonomic** - the computing system is capable of pursuing any prescribed goal (not just SLAs) in a *fully automated* way, thus completely relieving the human operator from system administration tasks.

Besides enabling a robust classification of self-managing systems based on their main attribute, autonomicity, the items in the list above may also be used for the qualitative evaluation of autonomic IT architectures in a comparable way (see 2.1.3).

Concretely, there exist several techniques [104] to implement self-management properties in practical systems. We provide a mapping of those techniques against the five levels of attainment introduced above:

- **Hot swapping** is a technique for *self-configuration* that injects monitoring and diagnostic routines into live code. This operates on the **predictive** level.

- **Data clustering**, a machine learning technique, is used to infer efficient registry settings, also a realisation of *self-configuration* at the **predictive** [105] and **adaptive** [24] levels.

- **Control theory** can potentially provide support for producing the desired system output, thus realising *self-optimisation* at an **adaptive** level [125].

- **Hardware and software redundancy** is a *self-healing* mechanism, allowing the replacement of faulty components with back-up ones. The back-up components may be dynamically created [20], thus placing this technique on the **adaptive** level of attainment. A variant of this approach is **component level rebooting** where the system unit affected by a failure rolls back to the latest known stable state and restarts. This places the system in the **basic** category.

- **Probes and sensors** are used to collect data about the system state and feed it into an engine that compares it against fault models and notifies system administrators about the result. This is a *self-healing* technique that attains a **managed** level of autonomicity.

- **Self certifying alerts** endorse *self-protection*. Specifically, the state of system components is periodically compared against known stable models. When anomalies are detected, their signatures are dispatched in the form of alerts to all other system components. A system equipped with this technique shows an **adaptive** level of autonomicity.

At any level of self-management attainment, we note the following advantages of introducing autonomic behaviour to IT systems [90]:

- the considerable reduction and potential elimination of tedious system administration labour performed by human operators;

- the reduction of the level of IT knowledge and skills required of human operators (ideally, the only human involvement with the autonomic process would be at policy level, for prescribing system goals and operation constraints, a task that only requires high level knowledge of the underlying architecture);

- the decrease in response time to critical situations, especially in environments where humans are unlikely to intervene in realtime (e.g., space exploration [180]);

- a potential standardisation of system management, provided that autonomic architectures continue to be developed under open standards (a comprehensive list of such standards, both well established and emerging, may be found in [90]).

2.1.1 Structure

An *autonomic element* (Fig. 1.1), namely a computing entity (hardware, software or a combination of the two) capable of some level of self-management, comprises an *autonomic manager* governed by *policies* and connected to a *managed resource* via *sensors* and *effectors*. Each of the autonomic element's components is presented in the following.

The Managed Resource, Sensors, Effectors

The *managed resource* is any hardware (e.g., a cluster of computers in a network, an array of physical devices in a pervasive computing environment such as a smart home, etc.) or software (e.g., an operating system, an electronic resource such as a digital document, a database or a knowledge graph, etc.) entity that is not (but should be) capable of self-management. The state of the managed resource is exposed to other components (such as the autonomic manager) by *sensors* [90], either embedded (performance monitors, native to the operating system, relaying information such as CPU speed or HDD capacity) or external (bespoke monitoring software capable of detecting complex changes, such as edits in a document). *Effectors* are defined in [90] as interfaces that allow changing the state of the managed resource, by tuning parameters such as room temperature in a smart home or changing the size of text in presentation slides.

Policies

All the operations triggered by monitored changes and eventually leading to the actions implemented via effectors are regulated by *policies*. These are "behavioural constraints" [90, 102] guiding the otherwise independent operation of an autonomic element. Their main advantage is that they allow de-coupling high level requirements from system implementation [87, 174, 11], such that the former could (ideally) be modified without rewriting the latter. Traditional policy classification [102] relies on artificial intelligence metaphors, namely "reflective agents", that merely implement the actions prescribed by their governing

policies, and "goal-oriented agents", capable of devising their own actions in order to attain a given goal. Drawing on that parallel, Kephart and Walsh identify two types of policies: *event-condition-action* (ECA) and *goal* policies (with their enhanced variant, *utility* policies). The three types of policies, the formal languages they are formulated in as well as policy conflicts are discussed in section 2.1.2.

The Autonomic Manager

The *autonomic manager* facilitates the realisation of self-* properties by implementing the MAPE-K intelligent control loop. The *monitor* phase is closely related to the operation of *sensors* whereas the *execute* one relies on *effectors*. Although displayed separately in Fig. 1.1, the *analyse* and *plan* stages do not necessarily have dedicated hardware or software modules to support them. On the contrary, analysis and planning tasks are usually tightly coupled and interleaved (e.g., during incremental plan development, additional analysis is carried out to inform the selection of each new plan step). A more detailed discussion about analysis, planning and knowledge within the autonomic control loop is presented in the following.

- *Analyse*. The role of this component is to create models from monitored data, match them against known patterns and decide whether a change is needed to drive the system to a state where the prescribed objectives may be achieved [90]. The actual implementation of the analysis phase is either vaguely described in the literature or omitted altogether.

- *Plan*. During this stage, the autonomic system constructs the (sequences of) actions needed to achieve high level goals [90]. Thus, planning may be viewed as a search problem over the space of all possible sequences of actions derived from the autonomic system's knowledge base [182, 88]. To address the challenge of automated planning in partially known environments, Srivastava et al. [165] suggest keeping records of previously executed plans and annotating them with metadata for potential reuse (giving rise to the idea of "plan life cycles").

- *Knowledge*. The operation of the autonomic control loop relies strongly on the knowledge that the manager stores about itself (known data patterns used to analyse sensor input, previously utilised plans, policies, etc.), the managed resource (legacy architectural models or some other form of expert knowledge) and the outside environment (sensor logs). In order for the analyse and plan phases to run smoothly, the knowledge repository needs to be kept up to date with the changes in the system. Also, the chosen knowledge representation should support fast processing (information retrieval and analysis/reasoning) at runtime. In light of these requirements, it is recommended [88] to represent knowledge as a collection of components linked by connectors, restricted by constraints and described in a bespoke Architectural Description Language (ADL), such as Darwin [110, 88], ACME [152] or xADL [50].

2.1.2 Policies

ECA policies exhaustively list all the actions that an autonomic system should take in a given context. For exemplification, consider the self-adaptive document rendering application introduced in section 1.3.1. A possible ECA would be:

When audience focus drops below the medium threshold	*event*
AND	
it is afternoon	*condition*
THEN	
increase font size and increase display brightness.	*action*

The scenario that this policy describes is that of a presentation scheduled in the afternoon, where the area of exposed iris, recorded by room cameras and averaged over all members of the audience, corresponds to a previously defined level (i.e., medium) of focus. If this event-condition combination is detected, the autonomic system should increase the size of all text and the brightness of the display. Note that ECA policies formulated as above virtually eliminate the need for the analyse and plan phases of the MAPE-K loop. The autonomic manager is made aware of the two triggers that should be reported by the monitoring system at the same time (a camera feed pointing to a medium level of focus and the system clock indicating a time in the PM range), thus making any further analysis unnecessary. The ECA policy also provides a fully developed plan, namely the exhaustive set of actions that need to be performed (increase font size and increase display brightness). This reduces the control loop to just two phases: monitor and execute. The knowledge (what to modify, namely font size and display brightness) required by this simplified model is also contained in the policy.

Unfortunately, ECA policies have a major disadvantage, in that they may contradict each other, especially when prescribed in large numbers (as is the case with most realistic autonomic systems). Let us illustrate such a conflict by referring to the second application, presented in section 1.3.2. An ECA policy enforcing the *correctness* of the knowledge base is:

When the knowledge graph is edited by a user	*event*
AND	
the author of the edit has a credibility score higher than 5	*condition 1*
AND	
all reasoner checks pass	*condition 2*
THEN	
commit change to live server.	*action*

This policy ensures that all modifications made to the knowledge base through the user interface are authored by reputable actors as well as logically consistent, as determined by the reasoner. Note that this policy does not provide a safeguard against semantic violations, for example, it will not reject an edit stating that GOOSE is a sub-career of PHYSICS, as long as it is consistently defined with respect to the ontology structure.

Another goal of the system is to offer the public a complete version of the careers ontology, one that captures the entire community's views on the matter. Hence, a *completeness* ECA policy is defined:

When a user requests to view the knowledge graph or one of its sections	*event*
THEN	
display all available information (nodes, connections, meta-data) relevant to the user query.	*action*

The conflict between the correctness and completeness policies is subtle. Some of the users with a credibility score under 5 are new to the system and still building a reputation as trust-worthy editors, therefore their low rank is not necessarily a consequence of (semantically) bad edits suggested in the past. However, their proposals will be rejected according to the correctness policy. This inherently hinders the execution of the completeness policy, as the displayed knowledge will not reflect the views of new users from the careers community.

Goal policies express the autonomic system's objective, in other words, the desirable state that the system is meant to reach. The sequence of actions to be executed in order to achieve the prescribed objective is devised by the system rather than explicitly provided, as in the case of ECA policies. For instance, the policy for the document rendering scenario defined above can be reformulated as a goal policy.

Maintain audience focus at a medium level or above.	*goal*

The correctness and completeness ECA policies for the careers network application can also be merged into one goal policy:

Maintain and display a correct and complete version of the knowledge graph.	*goal*

In order to reach the final state prescribed by the goal policy, the autonomic system will independently analyse its knowledge base and plan the necessary actions, namely increase text size and adjust brightness, within the confines of available battery levels, for the first scenario and accept/display logically correct edits from all users with the appropriate credibility score for the second.

The main advantage of performing analysis and planning as distinct stages of the MAPE-K loop, rather than delegating to ECA policies, is that policy authors no longer need to prescribe the exhaustive list of actions necessary to fulfil a goal, as the autonomic system is capable of performing that task on its own. However, increasing the degree of system autonomicity via goal policies does carry a significant disadvantage, as compiling the sequence of actions during the planning phase is more computationally intensive than directly retrieving it from the body of ECA policies. Furthermore, goal policies are still affected by conflicts. To exemplify, consider the following two goal policies for the document rendering application:

Maintain audience focus at a medium level or above.	*goal*
Maintain battery charge at a medium level or above.	*goal*

Modifying the way slides are rendered (increasing brightness, in particular) during a presentation in order to capture the audience's attention is likely to drain the battery at an accelerated rate. This makes the two policies above inherently conflicting. One way of resolving the contradiction is to assign a numerical value to each of the states described by the policies such that the system could unequivocally determine which one is more desirable.

Utility policies disambiguate system states by assigning a numerical score (utility) to each of them. In the example above, let us consider that a high/medium/low battery charge is associated to utility scores

1/2/3, whereas a low/medium/high level of audience focus is evaluated to scores 1/2/3 as well. In this case, the system attempts to maximise the focus-related utility and minimise the battery-related one, that is, maintain a high level of attention without taking up too much power. If that is not achievable (e.g., due to hardware constraints), one of the remaining three combinations allowed by the policies will be selected: medium focus with high battery charge, high focus with medium battery charge or medium focus with medium battery charge. The advantage of this approach is that utility policies are inherently conflictless [102]. The disadvantage lies with mapping the space of system states to that of utilities, a task requiring in-depth familiarity with the managed resource and its environment [88].

Given that all previously discussed policies have both advantages and disadvantages, it is argued [102] that autonomic systems should use a combination of all three types in order to strike a balance between a high level of autonomicity and sustainable resource consumption. In this context, strategies for *deconflicting policies* become important. Although some research, see [116], recommends this task be externalised, that is, addressed by the issuing communities (of domain experts) prior to policy deployment onto the autonomic system, the main body of work in the field promotes the idea that conflict detection and, in some cases, resolution should be automatically performed, at runtime, by the autonomic system itself. In what concerns ECA policy conflict resolution, the following general approaches are suggested [102]:

- assigning priorities, a technique allowing the application of policies in order of importance, ultimately similar to the utility-based approach;

- adding meta-policies meant to disable the trigger of all conflicting policies save one; more specifically, the action part of the meta-policy represents the negation of the conflicting policy's condition;

- modifying the individual conflicting policies by transforming their condition into a conjunction of the existing one and the inverse of the conflicting policy's condition.

In practice, due to the intricacy of policy interactions, these approaches have limited applicability, thus reducing conflict detection/resolution to spotting and resolving logical contradictions [11] (e.g., the same action is both forbidden and required by different policies). This call for a bespoke formal language to support policy description and analysis. Some of the widely used *policy languages* and the way they support conflict detection/resolution are briefly discussed in the following.

The OWL-based Policy Language for Agent Reasoning, OWL-POLAR [155], describes policies in a format based on conjunctive formulae. Each policy has an activation condition, an action and an expiration condition. The actor executing the action as well as the action modality (obligatory, permitted or forbidden) are also specified. The example below shows how the ECA policy for the self-adaptive document rendering application would be formulated in OWL-POLAR:

When audience focus drops below the medium threshold	*event*
AND	
it is afternoon	*condition*
THEN	
increase font size and figure resolution by one unit each.	*action*
$Focus(?f) \wedge isBelow(?f, medium) \wedge Time(?t) \wedge in(?t, pm) \rightarrow$	*activation*
$O_{?e:Effector(?e)}?a_1, ?a_2 : increaseFont(?a_1) \wedge increaseResolution(?a_2) /$	*action*
$isAboveOrEqual(?f, medium)$	*expiration*

All formulae in the conjunction above are atomic assertions that either indicate a type - *Focus*, *Time*, *Effector*, *increaseFont*, *increaseResolution* - or represent a relation - *isBelow*, *in*, *isAboveOrEqual*. All the symbols preceded by a question mark are variables, whereas *medium* and *pm* are predefined constants. The expressions following a colon are constraints applied to the variables preceding it. Symbol *O* in the OWL-POLAR policy action represents the obligation modality, stating that the action following it is mandatory.

The supplementary effort of translating policies from natural language into conjunctive formulae is justified by the fact that OWL-POLAR policies are machine readable, thus can be automatically reasoned upon. To exemplify, [155] proposes two algorithms for policy reasoning: one for eliminating redundant policies (algorithm 1 in [155], based on a reasoning technique called subsumption) and one for anticipating policy conflicts (by utilising a type of reasoning called consistency checking, as illustrated by algorithm 2 in the same paper). Once detected, conflicts can be resolved by developing plans, at runtime, to activate the expiration conditions of all but one of the contradicting policies.

The Knowledgeable Agent-oriented System, KAoS [178, 28, 69], provides a graphical interface (the KAoS Policy Administration Tool, KPAT) to assist stakeholders with defining policies. This way, the translation of policies from natural language into DAML[1], the formal language used internally by KAoS, is hidden from policy authors, thus simplifying their task. KAoS utilises a collection of high level ontologies, featuring specialised concepts for all policy components such as actions, actors and required resources. These concepts are appropriately instantiated to represent bespoke policies, forming an application specific ontology (for example, in the case of the self-adaptive rendering application, instances *increaseFont* and *increaseResolution* would be created for concept *action*). The ontology is afterwards used to detect policy conflicts via consistency checking and to assign policy actions to specific actors (enforcers, in KAoS terminology). Policy deconfliction is performed with respect to pre-set priorities. Although the acronyms are similar, KAoS is a concrete implementation of a policy management system and is not to be confused with KAS, the template architecture for autonomic systems hybridised with semantic technologies, proposed in chapter 4.

Rei [94] is a language built on top of Prolog that represents policies in a fashion similar to OWL-POLAR. More specifically, Rei policies have actors, actions and modalities such as allowed, forbidden, compulsory and dispensed (equivalent to postponed obligation). Like KAoS, Rei employs domain independent as well as problem-specific ontologies to represent policy components and detect conflicts. The latter are resolved by means of meta-policies and pre-set priorities.

---

[1]http://www.daml.org/about.html

2.1.3 Applications and Evaluation

Autonomic computing applications rage from full system implementations [80, 95, 1, 182, 155, 27] to bespoke practical realisations of the MAPE-K components (monitor [2], plan [39], analyse [36, 6]). A detailed analysis of state of the art contributions is provided in 3.1, however, to paint a picture of the range and diversity of autonomic applications, the major problem domains where such technologies are applied are given below[2].

- space exploration: NASA ANTS [88] supports collaboration between asteroid belts probes;

- distributed systems: load balancing and component repair [95], data collection frequency control [2], dynamic resource management [1, 182, 150, 147], network compartmentalisation and assembly [27], network parameter estimation [131, 148, 6], medical services in the cloud [4];

- electronic personal assistants: PC performance improvement via operating system's processes management [108];

- web services configuration [124, 184];

- military and security-critical applications: hierarchical policy management [37];

- middleware [57, 117, 187];

- autonomic database management [3];

- data centre management [182].

Given the wide range and the increased diversity of autonomic applications, it is not surprising that a unified evaluation framework is yet to be formulated. The most common scenario is to test the performance of the autonomic system in the context of the application it was designed for. This does little in the way of enabling a rigorous performance comparison across a range of different systems. However, there exist a few proposals of general assessment criteria for autonomic systems that can be roughly classed as either qualitative or quantitative.

Qualitative Evaluation

Qualitative evaluation employs discrete, non-numeric scales for various system aspects [131]. IBM's five levels of autonomicity (2.1) represent such a scale, used both in [88] and as one of the classification criteria in 3.1. Other qualitative metrics suggested in [131, 127, 113] are:

- **architecture category**: flat (horizontal) or hierarchical (vertical, where managers on a given level are coordinated by one situated on the level above),

- **adaptation approach**: the strategy the system employs to adapt to changes in its environment (can be policy based or utility function based),

---

[2]There is significant overlap between contributions: some papers should, in all fairness, be included in several categories. This, as well as the abundance of evaluation criteria, makes it difficult to provide a clean classification of autonomic applications. A more thorough categorisation than that given here is attempted in chapter 3.

- **learning ability**: splits autonomic systems in two categories based on the presence or absence of a learning mechanism (there is a quantitative evaluation metric to capture **learning efficiency**, as shown in the next section),

- **openness**: autonomic services that are publicly available or with restricted access,

- **evolvability**: discriminates between autonomic systems that can be maintained and extended and those that cannot (a trademark of an evolvable system is the fact that its core elements are designed as middleware [131]),

- **granularity**: measures the modularity and coupling of system components [127] (either thick-grained or fine-grained),

- **robustness**: refers to a system's capacity of avoiding/ recovering from failures [127],

- **validation**: differentiates between systems assessed via simulation, mathematical modelling or other qualitative/quantitative metrics, on the one hand, and systems without any associated evaluation data, on the other hand.

Quantitative Evaluation

- **Probabilistic verification techniques** measure how well systems meet quality requirements, such as "the probability of a successful attack must be lower than a given threshold". The availability of a probabilistic model (e.g., based on Markov chains [34]) of the system is a prerequisite. This implies that system properties as well as quality requirements must be translated from informal language into a probabilistic temporal logic, with substantial computational effort [33]. ProProST [75], introduced to simplify the translation process, represents a collection of specification patterns to be used as building blocks and assembled to formulate arbitrarily complex probabilistic properties. A structured English grammar is also available to aid the translation of quality requirements and system properties from natural language into ProProST.

- **Application specific metrics** measure system performance in the context of the problem they were designed to solve. A list of such metrics [131] includes the **quality of the autonomic response** (e.g., the increase in the speed of response, the decrease in the overall number of faults over a given period of time, etc.), the **cost of autonomy** (the overhead introduced by the MAPE-K loop), the **flexibility ratio** (the rate of failure decrease divided by the overhead), the **speed of the autonomic response** (the total time necessary to adapt to change in the environment), the **degree of proactivity** (the number of unsupervised decisions made by the system).

- **Generic metrics** give an objective, application independent evaluation of (theoretically) any type of autonomic system. These metrics [131] may be relevant for comparing different autonomic systems on the condition that they were built to achieve similar goals. The quantitative evaluation criteria proposed in [131] have to do with the autonomic system's efficiency at storing and manipulating its knowledge base. The **coefficient of learning**, c_l, is meant to assess the system's capacity to learn from experience (stored in the knowledge base) in order to enhance future performance and is formally defined as follows:

31

$$c_l = \frac{max(1 - \frac{E\{D'\}}{E\{ND'\}}, 0) + min(\frac{E\{D\}}{E\{PD\}}, 1)}{2}, \tag{2.1}$$

where $E\{D\}$ represents the average number of correct learning based decisions and $E\{PD\}$ stands for the average number of "target achievers", namely decisions with a significant impact on realising the system goal. $E\{D'\}$ and $E\{ND'\}$ are defined similarly, yet for incorrect decisions, with the latter representing the number of "target damagers", i.e., decisions with a strong negative influence on the system (driving it away from a desirable state). Systems successful at learning from their experience will score close to 1 on the c_l scale, meaning that out of all target achievers, most were learning based decisions, whereas out of all target damagers, almost none were learning based decisions. Systems inefficient at learning from their knowledge base will have a c_l value close to 0. The **efficiency of learning**, e_l gives a numeric interpretation to the amount of progress made by the system towards the target goal:

$$e_l = \frac{K\{PD\}}{K\{PD\} + K\{ND'\}}. \tag{2.2}$$

Symbols PD and ND have the same significance as above, whereas K stands for "average percentage". These last two metrics are used to define the **learning index**:

$$I_l = \frac{L}{M} \cdot c_l \cdot e_l, \tag{2.3}$$

where L represents the number of system parameters that can be learned and M stands for the total number of system management parameters. A system with a learning index of 0 either has no learnable parameters ($L = 0$) and/or is incapable of learning from knowledge ($c_l = 0$) and/or makes no target achieving decision ($e_l = 0$). In [131], such a system is termed *closed adaptive system*. *Open adaptive systems* are situated at the opposite end of the spectrum, with an I_l of 1. The **accuracy of awareness** is a metric targeting the frequency of changes in the system's knowledge base (preferably high, to reflect environment dynamics) as well as the overhead introduced by it (preferably low, to speed up execution). To assess the accuracy of awareness, [131] suggests taking several measurements including the *time* required to insert a new data sample, the *trust* of the update's author (e.g., a credibility/reputation score) and the *correctness* of the knowledge base (the number of other data samples corroborating the newly added one).

2.1.4 Reflection and Open Issues

The foundational concepts and techniques around autonomic systems' purpose, structure, applications and evaluation give rise to a series of reflection points discussed in the following.

Self-* properties are revolutionary: they represent a complete change in perspective relative to classical system architectures and have the *potential* of unlocking genuinely autonomic behaviour in practical contexts. The main reason why this has not yet happened in reality is that self-* properties are also very abstract [90, 101], causing researchers to implement their own interpretation and ultimately produce fit-for-purpose autonomic instances that are "locked" inside a particular application domain. There are very few documented attempts at *understanding* self-* properties (e.g., the conceptual parallel

to the human ANS [90]) and *implementing* them in an architected, reproducible manner (some high level techniques are available [104]). To make autonomic systems more extensible and portable across problem domains, there is a need for **design guidelines as well as good practices and techniques for implementing such systems** (programming models [5]).

Standardised metrics to evaluate and *compare* autonomic systems' performance are in short supply [88, 5]. Some qualitative and quantitative evaluation techniques are available (2.1.3), but they are difficult to retrofit to existing systems given that the elements that should be assessed are not described in sufficient detail. For instance, the coefficient of learning (2.1) requires the number of learning based decisions that were either correct or incorrect as well as their impact on the system's goal, details that are unavailable for most autonomic systems in the literature, ultimately making it impossible to use c_l as a benchmark metric. Authors tend to evaluate their systems by deploying them onto managed resources and measuring the degree to which their bespoke application goal was met, an approach that does not hold outside a narrow problem domain. A case for **benchmarks for measuring self-* properties** is made [103, 150].

Reusable autonomic components will be integrated in larger applications, where other modules will rely on their capability to achieve pre-set goals. This brings about the need for *trust modules* capturing both the *reliability* of autonomic components as well as their *adaptability* to change [88]. This links to the previous reflection point as it calls for **evaluation criteria** to measure these two features.

The *nonlinearity of emergent behaviour* [101] makes it difficult to predict how the interaction between several autonomic elements will influence the performance of the entire system. Kephart and Chess identify this as the main obstacle in the way of achieving truly autonomic behaviour (level 5 on IBM's autonomicity scale). They approach the problem in a top-down manner and suggest the development of a powerful modelling tool (e.g., an extension of distributed and hierarchical control theory [103]) to model the behaviour of complex autonomic systems. Contrastingly, this work takes a bottom-up approach, namely modelling individual autonomic elements using existing formalisms (state hierarchies, DL, knowledge graphs) and then **using the individual models to guide integration**. This is significantly simplified by functionally decoupling co-dependant MAPE stages, such as analysis and planning.

Knowledge about system components and services should be stored inside the autonomic manager and represented in a **machine readable formalism** that is expressive enough to allow for reasoning [5]. This requirement becomes essential in the case of systems guided by ECA policies [88], since the condition part implies investigating the system state and the action part entails modifying it. Therefore the system state should be both accessible and programmable. This directly impacts the efficiency of planning (plans being explicitly contained within ECA policies).

There is a need for *open access, extensible autonomic tools* veering away from the configuration issues of heavyweight solutions such as ACT [150]. Architectures based on **lightweight components** that can be used as building blocks elsewhere and fully disclosed in terms of implementation would represent a significant step in that direction.

2.2 Semantic Technologies

The majority of today's Web is organised as a series of documents that are *displayed by software* (text font and size are encoded in HTML/CSS scripts that are interpreted by browsers to control the appearance of online content). However, interpreting the *meaning* of Web information (determining whether a piece of text is a credit card number or a hotel review) is left exclusively to *human readers*. This becomes particularly inconvenient in situations where data from various sources (and in different formats) needs to be considered in order to reach a decision. For instance, when booking a holiday online, the Web 1.0 user needs to compile information from air carrier, hotel and tourist guide websites in order to match the available offers against personal preferences and constraints and finally select the most suitable option. What is, in the context of holiday booking, just a minor nuisance, escalates to a full scale, costly problem in a business environment. On a daily basis, companies process unstructured information originating from emails, news outlets, internal reports, legal documents, third party product specifications, etc., all in different formats/styles and from separate sources/authors. Given the harsh time constraints placed upon corporate decision making, the considerable effort behind analysing and integrating all that data is unsuitable for (a team of) human experts. The direct consequence is that businesses have to base important decisions on incomplete, possibly misunderstood knowledge.

One (partial) solution to the problem is the emergence of Web 2.0 [22, 45]. This new iteration promotes user centric applications (YouTube, Delicious, Flikr) where human Web explorers add tags to online resources as a by-product of social pursuits (publishing, sharing and commenting on videos, web links and photos), a process referred to as "social tagging" [73]. Over time, user metadata builds into rich contextual descriptions (e.g., the set of tags assigned to a Flikr photo by different viewers) enabling *machine processing* over the space of Web 2.0 resources (one example is the algorithm that searches for Flickr photos based on the popularity of their tags). However, Web 2.0 tags are not regulated by a common vocabulary, encouraging tag synonymy (user A tags a photo with `cinema` and user B tags the same photo with `movie_theatre`, with no way for a search algorithm to infer that the two annotations are equivalent). Under these circumstances, the entirety of user tags for a given collection of Web 2.0 resources is called a *folksonomy* [179, 93].

Semantic technologies address that limitation by extracting annotations from shared repositories of knowledge called *domain ontologies* [19, 73]. Written in a machine readable, logic-based language (XML, OWL, etc.), ontologies store hierarchies of concepts (along with their properties) representing physical entities from the modelled environment. This way, ontology triples `<cinema is_a projection_house>` and `<movie_theatre is_a projection_house>` will be used to *infer* that annotations `cinema` and `movie_theatre` are equivalent. Also, should a user run a semantic search for `cinema`, the result list would contain resources annotated with `cinema` as well as resources annotated with `movie_theatre` (along with all other related annotations stored in the domain ontology). This process is called *approximate querying* [44] and can only be performed in the Sematic Web (Web 1.0 search engines will only return exact `cinema` matches).

Besides avoiding the synonymity problem of folksonomies, ontology extracted tags allow computer applications to perform knowledge management in lieu of the user [40], with the obvious benefits of speed and convenience. In order to illustrate ontology driven semantic annotation and give some examples of powerful computing algorithms that can be built on top of the resulting tags, let us consider

the `http://nasajobs.nasa.gov/` website.

- An expert builds a domain ontology modelling career fields along with their hierarchical relationships such as "ENGINEERING is a sub-field of TECHNICAL SCIENCE" and "SPACE ENGINEERING is a sub-field of ENGINEERING ". Other related data, for instance "ENGINEERING has JACS G160[3]", where JACS G160 is the HESA code associated to the engineering field, is also represented.

- The text content of the webpage is separated in words that are matched against ontology concepts.

- Every word on the website is annotated with the matching ontology concept. Specifically, every occurrence of "space engineering" will be tagged with ontology concept SPACE ENGINEERING. This makes it possible to infer , based on the relevant ontology assertions, that the associated JACS value is G160.

A Semantic Web application that could be built on top of the semantically tagged content of `http://nasajobs.nasa.gov/` would extract the JACS data from the website annotations, match it against university programmes and produce a list of courses that a potential candidate would need to graduate from in order to be able to apply for a job at NASA. That report would be difficult to produce manually. Improved search over annotated resources [53] is another Semantic Web application example. This outperforms Web 1.0 search in three ways:

- a more comprehensive list of results is returned (given that synonym tags, extracted from the equivalence relationships stored in the domain ontology, are considered as well)

- each result is accompanied by a justification showing the semantic properties linking it to the initial query

- a list (or knowledge graph, as is the case with Google) of recommendations [141] is also produced, containing additional topics, semantically related to the initial query, that the user may be interested in exploring.

Ultimately, semantic metadata allows search engines to capitalise on the *context* that knowledge appears in, thus enriching the results list and presenting it in a more intuitive, connected way. To illustrate, the reader is invited to compare the results provided by Swoogle for search term `engineering`[4] against the ones returned by Google.

2.2.1 Core Advantages

In light of the presented overview, the following highlights of semantic technologies emerge.

Better exploitation of online knowledge. Annotating web resources with semantic tags extracted from ontology concepts supports the development of more powerful knowledge processing applications capable of providing more insight into the data they analyse. This ultimately "maximises the value" of information, as noted in [73]. For example, a semantic application tasked with planning a holiday is able

---

[3]https://www.hesa.ac.uk/component/content/article?id=1787

[4]`http://swoogle.umbc.edu/index.php?option=com_frontpage&service=search&queryType=search_swt&searchStart=1&searchString=engineering`

to process data with heterogeneous formats [143] (e.g., weather forecasts, hotel reviews and currency exchange reports), by analysing the associated tags rather than the data itself. Moreover, a semantic application is capable to *infer new information* [93] from the analysed data, for instance, the best time of year to book into a specific hotel, based on online reviews (see [76] for a detailed discussion on the benefits of semantic inference in another application domain, namely software engineering).

Integration. Just like in the case of autonomic systems, adding semantic support to an existing application does not require extensive re-engineering of legacy resources. On the contrary, the semantic layer (comprising domain ontologies, semantic annotations and middleware such as semantic reasoners and search tools) is sufficiently lightweight to wrap around the existing data infrastructure with minimum integration efforts [22].

Open standards. The World Wide Web Consortium (W3C) publishes and maintains a collection of recommendations[5] for linked data (ontologies) management, vocabulary languages usage (RDF, OWL, XML) and good inference/querying practices as well as guidelines for developing "vertical" applications connecting all layers of the semantic stack (2.2.2). This way, the integration of semantic support in existing applications is regulated by a central authority (the W3C) at no monetary cost[6] for the business.

Extensibility and cooperation. A domain entity is modelled within an ontology by an uniquely identifiable concept, that is, the name of each concept is prefixed with a bespoke URI [76]. This way, several ontologies can be merged/integrated in the same application [141] without needing to disambiguate overlapping terms.

2.2.2   The Semantic Web Stack

Semantic technologies are designed to support the Semantic Web, thus the layered diagram presented in Fig. 2.1 is a fitting visual support for their description.

Knowledge identifiers and representation languages

These are the first three layers of the stack, containing the specifications for URI (the prefix that makes every name in the ontology unique), XML and RDF (the languages that ontology concepts and properties are expressed in). These are considered to be well defined and complete semantic contributions with the widest industrial and academic acceptance [22, 45], relative to the other layers of the stack. The main reason behind the popularity of these identifiers/languages is that they are expressive enough to describe complex entities and relationships, yet, at the same time, sufficiently flexible to hide that complexity from the end user [121].

Taxonomies and ontologies

This level hosts the semantic models developed using the languages from the bottom layers of the stack. More accurately put, ontologies, taxonomies and rules are written in higher level languages (ultimately based on RDF/XML but with a simpler, more flexible syntax). Thus, the W3C recommended standard ontology language is OWL, whereas taxonomies are usually developed in RDFS (RDF Schema - a

---

[5]https://www.w3.org/standards/

[6]For access to standards and documentation - actual development costs are not considered here.
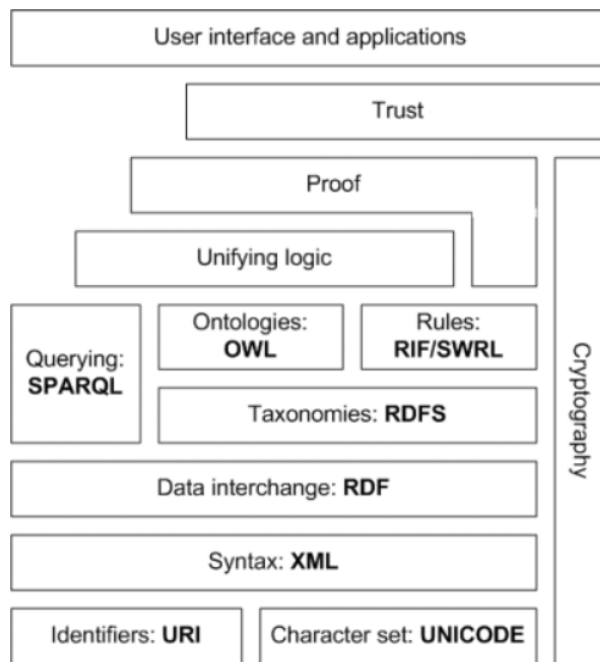
Fig. 2.1: The Semantic Web stack adapted from [84]

restricted set of RDF classes) and rules are build in either RIF (Rule Interchange Format) or SWRL (Semantic Web Rule Language).

Apart from usually being defined in different languages, ontologies, taxonomies[7] and rules [85] are difficult to separate (taxonomies can be viewed as simplified ontologies where the only relationship between concepts is hierarchical, whereas rules[8] are more frequently formulated inside ontologies than as stand-alone collections). This conclusion is easily reached by contemplating the multitude of (relatively vague) definitions offered in the literature for the ontology concept alone. To illustrate, a few of the most representative such definitions are listed in the following (the new ontology features introduced by each definition are italicised).

- An ontology is a *formal* and *shared* representation of a knowledge domain [158].

- Ontologies are *consensual models* of domains of discourse implemented as formal definitions of the *relevant* conceptual entities [77].

- Ontologies represent *explicit*, formal (as in machine readable) specifications of domain terms and their relationships, generating a common vocabulary for people/businesses sharing knowledge across the web [136, 143].

- Ontologies model both domain entities and *tasks*, namely the operations that are legal/allowed in the domain [65].

Summarising the above definitions, the research community views ontologies as formal, explicit, consensual and shared models of knowledge. Essentially, this means ontologies are self-contained,

---

[7]Although taxonomies are outside the scope of this work, the interested reader may consult [173] - slides 8 and 9 - for an excellent comparison between taxonomies and ontologies in a military application context.

[8]It has been proposed [84] to reorganise the stack into two towers to better accommodate a more robust framework to define and manage rules.

written in some machine readable language, agreed upon by the wider community and shared amongst its members. These key features can be treated as general design guidelines to support the ontology engineering process (more detailed design patterns are presented in section 2.2.4). A closer look at the final two definitions reveals what ontologies should contain, namely representations of domain terms (entities) and their relationships. This highlights another subtle connection between ontologies and autonomic systems, in the sense that ADLs (2.1.1), used to model the latter, describe components (which can be viewed as terms) and connectors (an alternative name for relationships). The final definition also mentions including system tasks in the ontology, that is, the operations that may be performed on the entities and their relationships. In an autonomic context, this would translate into storing plan actions and system goals in the knowledge base, such that they can be reasoned upon and potentially improved.

Querying, proof and trust

Semantic queries are expressed in SPARQL (Simple Protocol and RDF Query Language) [142], a language similar to SQL, capable of returning both explicit and implicit matches as well as providing a justification for the returned results. The justification will contain the axioms that were used to find the ontology concepts matching the search topic.

To illustrate, let us consider the holiday planning application described previously. Assuming that the user wants to spend his/her vacation in East Sussex, UK, one of the tasks of the planning application might be to compile a list of all hotels in that area. Running the SPARQL query in Fig. 2.2 will return all East Sussex hotels in the (fictional) `leisure_venues` ontology. Prefix `lsr` is associated to the ontology URI to be able to refer to ontology elements in a syntactically convenient way. The `SELECT` clause will search through all ontology concepts, find the ones that are the subject of the two triples in the `WHERE` clause, bind them to variable `?hotel` and return them to the calling code. The `WHERE` clause contains two typical RDF triples, where `?hotel` is the subject (the data the query is meant to return), `isIn` and `isA` represent the predicates, whereas `Hotel` and `EastSussex` are the objects.

```
PREFIX lsr: <http://xmlns.com/leisure_venues>
SELECT ?hotel
WHERE {
  ?hotel lsr:isA lsr:Hotel .
  ?hotel lsr:isIn lsr:EastSussex .
}
```

Fig. 2.2: A SPARQL query to find East Sussex hotels

The result generated by running the query will contain a list of hotels, each with the appropriate justification, similar to the one in Fig. 2.3 - note that none of the three ontology axioms is an exact match to the second triple in the query's `WHERE` clause. The fact that the returned hotel is in East Sussex is not explicitly stated, but inferred from the last two axioms in Fig. 2.3. If `leisure_venues` were a relational database instead of an ontology, this infered result could not be returned without significantly complicating the syntax of the employed SQL query (using foreign keys to join several tables) [129].

The same mechanism also supports semantic proof, namely the automated logical (in)validation of statements such as "BrightonCentral hotel has an occupancy of 70% in the summer". This is a powerful

```
BrightonCentralHotel

lsr:BrightonCentralHotel lsr:isA lsr:Hotel
lsr:BrightonCentralHotel lsr:isIn lsr:Brighton
lsr:Brighton lsr:isIn lsr:EastSussex
```

Fig. 2.3: One entry in the SPARQL query result

feature (unavailable, at least not straightforwardly, for systems storing knowledge in relational databases) that allows other applications to trust the validity of the results produced by semantic queries. Moreover, human end users are also encouraged to trust the outcome of semantic search, by being provided with a justification for each returned result.

However, semantic proof is not available for all ontologies. Ultimately, the more powerful the ontology language, the looser the guarantees that reasoning over it will produce trustworthy results [19]. For instance, OWL Lite, a simpler yet decidable language in the OWL family, will be able to guarantee the (in)validity of the statement about the summer occupancy of the Brighton hotel, whereas OWL Full, which is more expressive yet undecidable, will only be able to provide partial proof.

Unifying logic, cryptography and user-oriented applications

The top layers of the stack contain open challenges that the interested community, both academic and commercial, is still investigating (see 3.2). A representative example is cryptography, an ongoing, still unstandardised pursuit of semantic researchers and practitioners alike. Despite the importance of producing bespoke and efficient encryption algorithms to protect semantic data at all levels of the stack, while still allowing it to be searched and inferred over by client applications, the development of this component is still in its infancy [15]. This has a negative impact on the trust placed in semantic technologies by human end users and other applications.

The unifying logic refers to middleware meant to bind together the knowledge representation layers and the semantic querying component supporting inference and proof. Again, this is an open field, not currently regulated by any W3C standards, where researchers/practitioners create application-specific interfacing algorithms with little applicability outside the domain they were created for (3.2). This leads to a very fragmented top layer of the semantic stack, hosting heterogeneous client-facing applications (analysed in the state of the art chapter) optimised to successfully carry out a specific task with little scope for portability across problem domains.

A critical afterthought about the Semantic Stack

Given that the purpose of this work is to investigate ways to support autonomic behaviour by making use of semantic technologies, the knowledge representation and querying layers of the stack are of particular interest (2.2.4). Hence, a more detailed discussion about reasoning services (underpinning semantic queries amongst other functionalities) is presented in 2.2.3.

In a broader view, the core contributions of this research fit best in the user interface and applications layer. Specifically, chapter 4 proposes a general architecture and methodology to bind

semantic platforms (ontologies) and tools (reasoners, knowledge extractors) together with autonomic components in a singular framework designed to exploit the benefits of both technologies. The algorithm presented in Table 4.5 represents an implementation of that framework. To demonstrate how the architecture and methodology apply to different problem domains, chapters 5 and 6 describe two separate implementations, situated at the confluence between the semantic stack's applications layer and the practical segment of the autonomic computing domain.

2.2.3   Semantic Reasoning Services

Reasoning in DL [83] is a multi-faceted semantic service that may be broken down into five important functions.

Subsumption

Based on existing ontology axioms, semantic reasoners may infer knowledge that is not explicitly stated, but follows logically from previous assertions. For instance, let us consider the definitions[9] of ontology concepts `FrenchRivieraCity`, `NorthernMediterraneanCity` and `NorthernMediterraneanCountry` (Fig. 2.4). Based on the fact that `France` is among the countries to the north of the Mediterranean Sea, a semantic reasoner will infer that concept `FrenchRivieraCity` *subsumes* (is a subclass of) `NorthernMediterraneanCity`. This automated inference would prove very useful in the context of a travel planning application: should the user request a list of all vacation destinations north of the Mediterranean Sea, the results list would contain all cities on the French Riviera, including those that are not directly listed as Mediterranean cities.

```
Class: FrenchRivieraCity EquivalentTo:
  VacationDestination
  and (isLocatedIn value France)
```

```
Class: NorthernMediterraneanCity EquivalentTo:
  VacationDestination
  and (isLocatedIn some NorthernMediterraneanCountry)
```

```
Class: NorthernMediterraneanCountry EquivalentTo:
  {Spain, France, Monaco, Italy, Slovenia, Croatia, Bosnia and Herzegovina,
  Montenegro, Albania, Greece, Turkey}
```

Fig. 2.4: Concept definitions from an ontology describing vacation destinations - based on these, the reasoner subsumes `FrenchRivieraCity` under `NorthernMediterraneanCity`

Subsumption simplifies the ontology design process, allowing the ontologist to focus on formulating accurate class and property definitions rather that investing effort in determining the correct place in the ontology hierarchy where a new concept should be inserted.

---

[9]All definitions of ontology concepts and properties are given in the Manchester OWL syntax, available at `https://protegewiki.stanford.edu/wiki/Manchester_OWL_Syntax`.

Satisfiability

A common problem with large ontologies is that subtle, usually unwanted and difficult to detect connections tend to appear between axioms. In this context, it is possible for a concept to become *inconsistent* (or *unsatisfiable*), meaning that no instances could be created from it. The example in Fig. 2.5 illustrates such a scenario: an `ItalianRivieraCity` is both an `EuropeanCity` and a `NorthernMediterraneanCity`, however, the touristic appeal of European cities, on the one hand, and northern Mediterranean cities, on the other hand, is reflected by different average ranks[10]: 3 and, respectively, 5. Given that `hasAverageRank` is a functional property, it follows that Italian Riviera cities have two average ranks, which is both conuterintuitive and logically incorrect. The reasoner will address this situation by flagging concept `ItalianRivieraCity` as inconsistent.

```
Class: ItalianRivieraCity SubClassOf:
  EuropeanCity and NorthernMediterraneanCity
```

```
Class: EuropeanCity EquivalentTo:
  VacationDestination
  and (hasAverageRank value "3"^^integer)
```

```
Class: NorthernMediterraneanCity EquivalentTo:
  VacationDestination
  and (hasAverageRank value "5"^^integer)
```

Fig. 2.5: Concept definitions from an ontology describing vacation destinations - based on these, the reasoner infers that concept `ItalianRivieraCity` is unsatisfiable

Satisfiability checks are particularly useful during ontology design or when porting legacy databases to a semantic format, as they expose design flaws, usually too subtle for human domain experts to detect. Relative to the `ItalianRivieraCity` example, the ontologist is made aware that `hasAverageRank` causes an inconsistency, prompting either a redefinition of the culprit property, or, better yet, a reconfiguration of the concept hierarchy.

Synonymy

Ontology classes that have the exact same set of instances are called equivalent, a relationship that is automatically detected by the reasoner. In some cases, synonymy is useful: for instance, in the medical domain, determining that several terms refer to the same condition may support doctors with prescribing the correct treatment. In other cases, equivalent classes signal a design flaw, such as ambiguous definitions or insufficiently constrained axioms. The latter case is illustrated in Fig. 2.6, where concepts `EuropeanCity` and `AsianCity` lack the `isLocatedIn` property, therefore the reasoner will infer they are equivalent. Moreover, concept `City` is incorrectly defined as the intersection, rather than the reunion of the five continents, leading the reasoner to infer that `City` subsumes `EuropeanCity` and `AsianCity`. As the latter two concepts are asserted as subclasses of `City`, a cycle is formed. From

---

[10]Calculated, for instance, based on feedback left by travel websites' users.

a logical point of view, a cycle is not an inconsistency, therefore the reasoner does not flag such cases as problems. However, in some situations, as is the one illustrated in Fig. 2.6, a cycle may mask a design flaw. Specifically, the fact that concepts `City`, `EuropeanCity` and `AsianCity` are reported as synonyms will allert the ontologist to the error in the definition of concept `City` (where and should be replaced by or) as well as the ambiguity in the definitions of the other two concepts.

```
Class: EuropeanCity EquivalentTo:
  City
  and (hasAverageRank value "3"^^integer)
```

```
Class: AsianCity EquivalentTo:
 City
 and (hasAverageRank value "3"^^integer)
```

```
Class: City EquivalentTo:
  EuropeanCity and AsianCity and AmericanCity
  and AfricanCity and AustralianCity
```

Fig. 2.6: Concept definitions from an ontology describing vacation destinations - based on these, the reasoner infers that concepts `EuropeanCity`, `AsianCity` and `City` are synonyms

Query Answering

Provided that the underlying ontology is consistent, semantic reasoners are capable of automatically answering queries based on both asserted and inferred knowledge. Relative to the example in Fig. 2.4, asking whether France is a northern Mediterranean country represents a query of the former type, whereas requesting the list of all northern Mediterranean cities would fall under the latter category. Queries based on inferred knowledge are particularly powerful, allowing entries such as `Nice`, defined in Fig. 2.7, to be included in the results list, even though there is no explicit ontology axiom directly linking them to `NorthernMediterraneanCity`, the only search criterion mentioned in the query.

```
Individual: Nice
Types: FrenchRivieraCity
```

Fig. 2.7: Individual definition from an ontology describing vacation destinations - based on this and the concepts in Fig. 2.4, the reasoner infers that `Nice` is a northern Mediterranean city

The practical applications of answering queries based on inferred knowledge (ranging from allowing Google to provide more relevant results to supporting Siri with better understanding questions) come at a computational cost. Specifically, in order to determine that Nice is a northern Mediterranean city, that statement needs to be verified for all states of the modelled domain that satisfy the ontology's axioms. In the example given here, that is not an intense computation, however, realistic ontologies are of a much larger size. The complexity of semantic query answering can be traced back to the Open World Assumption (OWA), according to which facts that are not explicitly asserted are not assumed

to be false (as is the case with relational databases). Consequently, there is a compromise to be made between the flexibility enabled by OWA - the only realistic option available when modelling uncertain domains where missing knowledge is a given - and the effort entailed by computing entailments (query answers), a process close in nature to theorem proving [83]. In contract, query answering is significantly faster with relational databases, given that they operate under the Closed World Assumption (CWA). Regardless, modern reasoner implementations, such as FaCT++ [175] and Pellet [161], are capable of resolving ontological queries in a realistic time frame.

Inference Explaining

A fundamental service provided by semantic reasoners, tightly connected to the trust level of the Semantic Stack, is that of explaining inferences. Every semantic operation, particularly queries with counter-intuitive or unexpected results and computations that uncover design problems, such as concept inconsistency, is justified by a reasoning chain. For instance, when inquiring whether Nice is a north Mediterranean city, all axioms that support the result, namely the ones shown in Fig. 2.4 and Fig. 2.7, will be available for inspection upon producing the answer. Revealing the reasoning chain supporting a query answer is particularly useful when the ontology underpins an exploratory system, such as a career management application. For instance, users searching for the academic qualifications necessary to get a job in Physics, may come across other, related professional fields among the axioms in the reasoning chain and thus expand their range of job options.

In case of detecting unsatisfiable classes, the explanation is akin to the debugger output for code written in some programming language, in the sense that the ontologist can use the axiom trace to locate and address design flaws.

2.2.4 Ontology Engineering

Ontology engineering (OE) is an umbrella term [158] that comprises:

- *operations*, such as ontology design, extraction, querying and maintenance

- *methodologies*, namely a set of principles, patterns and strategies employed while performing the operations above

- *tools* (reasoners, knowledge extractors, etc.) and *languages* (OWL, SPARQL, etc.) supporting operation execution.

A closely related term is *ontology life cycle* [136, 135] that, in addition to the three OE facets above, also refers to ontology *evaluation* (2.1.3) and ontology *reuse* across problem domains.

The remainder of this section is structured with respect to OE operations, however, the focus lies with analysing the underlying methodologies. By doing that, it becomes possible to identify the design patterns/principles applied to build successful (expressive, maintainable, easily searchable, etc.) ontologies and reuse them in different problem domains. Moreover, methodology-based ontologies can be compared (structure and performance-wise), whereas bespoke ones, built without referring to design patterns or known good strategies, can only be evaluated in the context of the application they were designed for.

Ontology Design

Formally, a DL (description logic) ontology contains terminological (general) knowledge about the modelled domain, also known as the `TBox`, and assertional (specific) knowledge, referred to as the `ABox` [19]. The `TBox` contains classes (concepts or terms) describing generic domain entities (such as `Hotel` in the holiday booking example used previously), whereas the `ABox` comprises instances (individuals) of those classes (e.g., `BrightonCentral`). Selecting the structure of the ontology, namely the depth of the concept and instance hierarchies as well as the relationships (properties) that connect them, is performed during the design phase of OE.

Besides business rules, that impose restrictions related to the goal of the application hosting the ontology and the integration with other components, the structure selection process is informed by design patterns. These are known successful conceptual solutions to recurrent modelling problems, in other words, modelling "best practices" [65]. The usefulness of applying design patterns to solve any type of modelling problem is fairly intuitive, however, qualitative evidence is available [25] to support the claim that design patterns have a positive impact on ontology design (specifically, improve domain coverage, increase reusability and simplify maintenance). A list of commonly used design patterns and good practices (also applied while designing the ontologies for the systems presented in chapters 5 and 6) is included below.

Instantiation. Not all ontology concepts need be instantiated [136]. Although this may raise questions about the need for a class with no individuals, there are two main advantages to take into consideration. Firstly, "abstract" concepts (that do not model a physical entity in the problem domain) are sometimes necessary to represent complex properties that cannot be directly expressed as RDF triples (see the reification design principle for more details). These abstract concepts have no individuals. Secondly, reasoning over instances is still experimental, therefore semantic querying (ultimately performed by reasoners) is not as powerful on individuals as it is on concepts [19]. It thus makes sense to model specific domain entities as ontology terms if searching is an important feature of the host application.

Synonyms. If possible, it is preferable to define synonyms rather than equivalent classes [136, 143]. For instance, in the holiday booking ontology, `EastSussex` would be an instance of concept `County`, which can also be referred to as `Region`. Rather than asserting that the last two concepts are equivalent, `Region` could be made a synonym of `County`. Note that synonyms are values (in this case, strings) and are therefore treated as data type fillers during reasoning, thus, in all likelyhood, speeding up the querying process.

Cycles. Ontologies should not contain self-referencing concepts [136] (`Hotel` is a self-referencing concept if, for example, it is asserted that `Hotel` is a `HolidayFacility` and that `HolidayFacility` is a `Hotel`). Such cycles are not only logically redundant (or even inaccurate) but also impede (cause infinite loops in) the inference algorithms that semantic reasoning (therefore querying) is based on. Ontology cycles are difficult to detect and eliminate, since the self-referencing circle may span over many concepts (the provided example illustrates a simple case, where `Hotel` refers to itself via one other concept only). Cycle prevention at ontology learning time may prove computationally costly as it would require the knowledge extracting algorithm to check for self-referencing concepts as they are being added to the ontology. Cycle detection and elimination, however, is easier to perform after ontology learning has been

completed, as the reasoner will infer that all concepts in a self-referencing circle are equivalent (relative to the running example, triple `<Hotel isEquivalentTo HolidayFacility>` will be asserted). Thus, detecting cycles is reduced to retrieving and analysing the equivalence relationships in the ontology, which should not be many if the synonymy design pattern has been applied.

Reification. RDF triples relate a subject to an object via a predicate (in an ontology, the subject and object are concepts or individuals and the predicate is a property). This is not sufficient to model relationships between domain entities where the subject is related to multiple objects [166]. To illustrate, let us consider sentence "BrightonCentral has an occupancy of 20% in the winter and of 90% in the summer". Modelling that in an ontology implies asserting property `hasOccupancy` which is multi-faceted (has more than one filler, namely a percentage and a time of year). Reification addresses this issue by making use of an additional, "abstract" concept that will take over the fillers. Thus, the ontological translation for the previous sentence is the one in Fig. 2.8.

```
lsr:BrightonCentralHotel lsr:hasOccupancy lsr:OccupancyOne
lsr:OccupancyOne lsr:hasPercentageValue 20
lsr:OccupancyOne lsr:hasTimeOfYear "winter"

lsr:BrightonCentralHotel lsr:hasOccupancy lsr:OccupancyTwo
lsr:OccupancyTwo lsr:hasPercentageValue 90
lsr:Occupancytwo lsr:hasTimeOfYear "summer"
```

Fig. 2.8: A reified multi-faceted property

Libraries. Some researchers [65] view design patterns not as guidelines but as ontology building blocks (mini concept hierarchies known from experience to model a relevant knowledge domain well). Let us assume the domain being modelled is some text describing the physical realisation (book, file, etc.) of informational objects (poem, formula, story). One of the design patterns relevant to this scenario is a mini ontology where the concepts representing physical objects will inherit from `InformationRealization`, the informational ones will extend `InformationObject`, and the two hierarchies will be connected via property `realizes`. The authors of this approach provide libraries of content ontology design patterns that the ontologist need only import. The remaining task to complete the ontology is to create the leaf concepts. A strategy for selecting the best library pattern to import is also suggested. It implies running a comparison between the modelling problem specification (particularly the problem type description) and the design pattern intent (the type of problem it is designed to solve).

Ontology Extraction

Also known as ontology construction or learning, this OE phase implies asserting concepts and relationships that model the problem domain. These can be defined manually (by the ontologist, through an ontology IDE such as Protege), automatically (by an all-purpose, off-the-shelf or bespoke, application-specific ontology learning tool that extracts knowledge from a legacy repository) or by means of importing from another ontology [158, 143]. These approaches may be combined during the ontology learning process (automatically extracting the initial ontology from a legacy document and enriching it via manual edits is one of the options). Regardless of the chosen variant, the reasoner will automatically

place each new concept in the correct level of the concept hierarchy [143], a process called subsumption (detailed in the next section).

There are several methodologies proposed in the literature to guide the ontology learning process, depending on existing descriptions of the domain to be modelled:

- If there is no such prior description available (i.e., an ontology must be built to model a social or economic phenomenon that has not been previously observed), ontology learning is manually performed by a human domain expert.

- If a free or loosely structured text description of the domain to be modelled exists, such as manually written documentation for a piece of software, the ontology can be extracted by applying a machine learning approach (mainly, statistical text analysis) as explained below.

- If a well structured text description of the domain can be accessed (for instance, automatically produced software documentation, a standardised, computer generated report or even a manually created spreadsheet), then the format of the document can be exploited to build a custom ontology learning algorithm that is likely to run faster than machine learning based ones.

There is a broad body of research that analyses ontology extraction from free and loosely structured text [143, 158, 72]. The proposed methods are inspired from data mining:

- noun-verb analysis [136]: the nouns in the legacy document become ontology concepts, whereas the verbs will be made into ontology properties

- statistical analysis [143]: word occurrence frequency is calculated to determine the most likely candidates to become ontology concepts, word co-occurrence (how often two words appear together) is investigated to extract properties and clustering techniques are applied to detect synonyms

- natural language processing [143]: morphological and syntactic analysis is performed on the sentences in the text description, following that the parts of speech will be turned into ontology concepts, whereas their syntactic roles (subject, predicate or object) will determine ontology properties

The recommendation [143] to perform ontology learning from text in two phases (carry out shallow extraction from a large corpus of documents followed by in-depth analysis using aggregate statistics to refine the initial model) is particularly useful. The same approach can be adapted to extracting information from websites about careers : well known, reliable models about professions (HESA releases, JACS reports, etc.) can be analysed to build the shallow ontology that can be afterwards refined by human users exploring new online resources and extracting in-depth data. The advantages of outsourcing ontology enrichment to a community of interested users rather than a small group of domain experts are also noted in [77], where ontology learning is viewed as a social process constrained by technical bottlenecks, not the other way around.

To conclude this brief introduction to ontology learning, it is worth mentioning that this phase is reiterated all throughout the OE process. Intuitively, ontology learning can be viewed as "bootstrapping" [143], namely new, more complete and robust versions of the ontology will replace the old ones as the ontologist's understanding of the modelled domain improves.

Ontology Querying

Strictly speaking, querying is not a phase of OE, however, it is discussed here given its relationship with the underlying ontology structure. Semantic querying relies on inference, the process of deducing, by means of logical consequence, facts that are not explicitly stated in the knowledge model [40] (see the holiday booking example in 2.2.2). Inference can be straightforward or convoluted, depending on the the complexity of the ontology's design (redundant concepts/properties, storing equivalent classes in lieu of synonyms, a large number of individuals, etc. may slow down the process or even prevent it from completing). Thus, effective semantic query execution, such as the SPARQL example in 2.2.2, is tightly reliant on good, design pattern compliant ontology structure.

The connection between ontology design and semantic querying can be better highlighted by building a formal model of the inference process. Bayesian representations of explicit ontology relationships are used in [64] to produce a set of weighted if-then inference rules, a process called rule mining. A simplified version of the process can be illustrated on the following example (adapted from [64]): if 8 out of 10 instances of ontology concept `Manager` are connected via property `hasAge` to a number lower than 45 and via `hasProject` to an instance of concept `Project` featuring an `Increased` level of `Innovation`, then the extracted inference rule (with a weight of 80%) would be: "if the manager is under 45 then the innovation of the projects they are involved in is increased." Thus, rule mining supports the answering of queries such as "what is the degree of innovation of projects led by young managers", by drawing exclusively from the ontology structure (by means of *link analysis* to rank concepts based on connectivity and *frequent itemset mining* to search for patterns in instance definitions).

As pointed out previously, semantic querying retrieves *explicit and implicit* (by inference) knowledge from the web, thus satisfying the user query in a way that is unavailable in relational databases (that do not provide native support for hierarchical relationships [114]). To increase querying speed as well as maintain the broad coverage of the result set, [47] suggests combining ontology models with database storage provided the two share a set of common individuals. Association rules are derived between the two sources and are used to complete the knowledge in each of them based on related data from the other.

Besides the structure of the underlying ontology, semantic querying is also influenced by the way the user request is formulated. It is unrealistic to expect common web users to be capable of formulating a query in SPARQL (or another language directly interpretable by ontology reasoners). This brings forth the need to translate natural language into some sort of description logic language and also perform the opposite for the query results. This invites the use of tools and technologies from the field of natural language processing, ultimately introducing supplementary lags in the querying process [99]. Consequently, it is worth exploring alternative means of "exposing" the ontology knowledge to the end user, such as making use of intuitive graphical interfaces (6.3).

Ontology Maintenance

The two main facets of maintaining an ontology are:

- editing: modifying the ontology by adding, deleting or redefining concepts and properties to reflect changes in the modelled domain or in the interested community's understanding of it and

- alignment: merging several ontologies that describe overlapping domains, whilst resolving any redundancies and ambiguities.

In the case of ontologies modelling large, dynamic fields of knowledge (as is the careers one), centralised editing, that is, performed by the domain expert alone, is impractical and limiting. The mere size of the universe of discourse that the ontology is meant to represent makes in impractical to expect one expert (or a small group) to be aware of the need for as well as implement all the necessary modifications. Even if this were possible in a realistic scenario, the resulting ontology would represent the expert's view of the modelled domain, which contradicts the very definition of an ontology, namely a shared consensus about a particular field. Since a consensus cannot exist without a community, it is more feasible to have that community edit the ontology (in a responsible way) rather than forfeit that prerogative in favour of the domain expert. Of course, experts are not to be taken out of the process entirely. Instead, their knowledge can represent the first iteration of the ontology, after which they should migrate to the role of a curator, overseeing (and possibly validating) community edits.

Ontology alignment implies an additional level of meta-data, namely information about the modelled subject, hierarchy depth, etc., that is consulted when integrating separate yet compatible (describing semantically related domains) ontologies. Dictionaries are also necessary to eliminate conflicting definitions and address redundancies by asserting synonyms when appropriate [137]. Ontology alignment is a technically complicated problem and one of the open topics of semantic web research [45].

2.2.5 Applications and Evaluation

The acceptance of semantic web technologies is both industrial and academic, as confirmed by extensive surveys targeting both communities [45, 141]. To give an idea of the extent of that acceptance, the following is an enumeration of the main categories of semantic technology applications, accompanied by a few relevant examples. A more detailed continuation is provided in chapter 3.2 that focuses on ontology-related contributions.

- **knowledge management**: named entity recognition (matching news reel names with concepts from relevant ontologies) [45], automated financial content (generated by bank employees) processing (via referencing a backend ontology storing financial terms) [45], geospacial data management (by UK's national mapping agency, Ordnance Survey[11]), heterogeneously sourced corporate data aggregation (implemented by Oracle[12]), digital music repository construction (by the Norwegian National Broadcaster[13]), automated documentation generation for car repair and diagnosis (Renault[14]), subsystem mediation by means of an ontology modelling interfaces (BT[15]), software requirements engineering and insightful documentation generation (containing contextual data such as the places where variables are declared and used) [76], common vocabulary creation for the medical community (Unified Medical Language System ontology [26])

---

[11]https://www.ordnancesurvey.co.uk/

[12]http://docs.oracle.com/cd/B28359_01/appdev.111/b28397/sdo_rdf_concepts.htm

[13]https://www.w3.org/2001/sw/sweo/public/UseCases/NRK/

[14]https://www.w3.org/2001/sw/sweo/public/UseCases/Renault/

[15]https://www.w3.org/2001/sw/sweo/public/UseCases/BT/

- **semantic browsers and wiki pages**: Magpie [59, 13] allows the exploration of metadata (annotations) alongside the actual web resources, SweetWiki [30] provides a folksonomy that readers can use to annotate webpages as they browse and an interface for adding new concepts/relationships to the ontology

- **semantic search**[16]: Swoogle [53] (a semantic search engine that indexes semantic web documents and assesses their importance by computing ontology ranks), Watson [13, 48] (a more powerful semantic search engine complete with a set of APIs allowing applications to manage/integrate the knowledge that Watson finds), Search Thresher [17] (a Firefox plugin that discloses information about the reliability of the webpages in the search result list), CRUZAR[18] (the semantic tool for e-tourism developed by the city council of Zaragoza), PowerAqua [13] (a natural language question answering engine that runs Watson in the background), Scarlet [13] (a semantic search tool that runs Watson to identify relationships between query concepts across several ontologies), Xerox's FactSpotter[19], IBM's Omnifind[20]

- **natural language processing**: Boeing's BLUE [43], other promising attempts [121]

- **semantic web services and mobile applications**: meeting scheduling assistant based on calendar owner profile[21], mSpace Mobile [45] (a geographical locator that reads GPS coordinates and provides touristic and transport suggestions based on user preferences), BOTTARI [40] (an application that compiles social media posts to offer suggestions to users visiting a new, unfamiliar place)

- **middleware**: Semantic Web Framework [67] (features components for ontology engineering, customisation, querying, etc. to assist users with building their own semantic applications), NeON [168] (a family of nine methodologies for ontology engineering by reuse, a process described by Suarez et al. as "collaborative development of ontology networks"), Watson [48] (described by D'Aquin and Motta as a "complete infrastructure component for the development of applications of the Semantic Web", Watson provides APIs for searching ontologies, retrieving ontology metadata, computing ontology metrics such as concept density, executing SPARQL queries, word sense disambiguation, natural language question answering and others that can be used to build complex semantic applications

- **medical science**: Genome-Wide Association Study [40] (uses semantic tools to analyse raw biological data and rank marker-genes based on their relevance to a given condition), LarCK [40] (used by the World Health Organisation to match prior gene knowledge against new experimental data in cancer research), the Entity Describer [73] (collates several ontologies to give users a common vocabulary for annotating biomedical resources)

---

[16]Consult [22] for detailed and easy to follow examples of how semantic search returns relevant documents that classic search engines would not consider.

[17]https://www.w3.org/2001/sw/sweo/public/UseCases/Segala/

[18]https://www.w3.org/2001/sw/sweo/public/UseCases/Zaragoza-2/

[19]http://www.xerox.com/innovation/news-stories/text-mining/enus.html

[20]http://www-01.ibm.com/software/ecm/omnifind/

[21]https://www.cs.cmu.edu/ softagents/cal/

- **semantic tagging**: automatic recommendation of tags for annotating text [8] (calculates ambiguity and confidence scores for each candidate tag by matching it against a relevant ontology), Upper Tag Ontology [54] (collects Web 2.0 tags from Flikr, Delicious and Youtube in a centralised ontology), automated tag extraction via data mining [191], converting folksonomies into ontologies for tag disambiguation [179, 153] (via lexicographic stemming).

The examples above are aimed to illustrate the great variety of semantic technology applications. Given that, it is challenging to develop a coherent evaluation framework capable of measuring the efficiency of semantic systems across application domains. A notable attempt is that direction [45] analyses semantic tools by assessing their impact on both the research and industry communities. Two Hype Cycle Curves (Fig. 3 and Fig. 4 in [45]) are developed for this purpose, displaying the maturity and visibility of major semantic technologies as points on a graph. The curves represent the path from the technological trigger that launched a certain semantic technology until the plateau of productivity where the technology reached its practical potential. One of the conclusions drawn after comparing of the two curves is that researchers view semantic technologies as evenly spread between the hype curve's start and end points, with knowledge management solutions and standardisation contributions well within the productivity plateau. This contradicts the industry's view, where most semantic technologies are grouped around the early stages of acceptance (for instance standardisation is placed at the beginning of the slope of the enlightenment phase and the plateau of productivity is remarkably empty).

Given this misalignment between the views that different communities have with respect to the usefulness and progress of semantic technologies, it follows that *commitment* is a good metric to evaluate semantic tools' efficiency [77]. With respect to ontologies, commitment represents the number of users (or user communities) in agreement that the ontology *correctly* represents a *sufficiently large amount* of the modelled domain. Numerically, commitment is the ratio between the size of the ontology specification file (the one the ontology is extracted from) and the number of semantic web documents referring to the ontology [77]. The former is viewed as a measure of the level of ontology detail whereas the latter indicates the size of the community using the ontology.

In terms of ontology evaluation, apart from some radical views (e.g., stating that the quality of an ontology can be assessed *only* in the context of the application it is used for [136]) the techniques suggested in the literature mainly fall under two main categories, quantitative and qualitative, and are discussed in the following.

Quantitative Evaluation

- **Basic metrics**: class, parent and sibling counts are performed directly on the ontology in either a specialised editor (such as Protege [82]) or online[22] to assess the complexity of the ontology and the domain coverage it provides. To have these structural metrics capture some of the ontology semantics as well, a *normalisation* procedure is suggested [181], namely eliminating anonymous classes and properties, instantiating the leaf level concepts and running the reasoner to materialise the subsumption hierarchy.

- **Comparative metrics** [143]: ontologies are evaluated by comparison against gold standards

---

[22]http://mowl-power.cs.man.ac.uk:8080/metrics/

(known good ontologies constructed by domain experts) or different types of knowledge repositories (such as relational databases) covering the same domain.

- **Data driven evaluation** [79]: the accuracy of the ontology (how well it models the problem domain) is measured over the temporal and category dimensions. Specifically, the domain coverage provided by the ontology is analysed over time (as the modelled domain itself changes) and with respect to the different categories (e.g., business workflow, resource layer, etc.) of the modelled domain. The analysis implies decomposing both the domain corpus and the ontology corpus over a set of representative elements in a vector space and comparing them by means of cosine similarity. The process is applied to several open source ontologies and the statistical results show a significant variance in ontology coverage both over time and from one sub-category to another.

- **OE cost assessment**: ONTOCOM [157] evaluates the economical feasibility of the ontology engineering process by performing a parametric cost estimation for every OE stage (requirements analysis, conceptualisation, implementation, evaluation and documentation). The parameters (weights) are calibrated by OE experts.

- **Reasoner performance evaluation**: Instead of evaluating the complexity of the ontology classification procedure on the theoretical DL model (found to be NExpTIME-complete [96]), it is suggested to perform the calculations while taking into account the properties of the ontologies that the reasoners were deployed on. The prediction model found to best capture the performance of several OWL reasoners (FaCT++, HermiT, Pellet and TrOWL) was RandomForest [96] and it allowed the researchers to measure the impact that various ontology properties had on the complexity of reasoning. It was found that the number of existential quantification axioms in anonymous class expressions and the size of vocabulary had a strong influence, the number of cycles, the class in and out degree (namely, the number of incoming and outgoing properties) and the depth of inheritance had a medium impact, whereas the expression richness (the ratio between the number of anonymous classes and the total number of class expressions), and the number of equivalent properties had little to no influence. This is a discovery of high practical value as it states which properties should be measured and which should be ignored in order to estimate, with good accuracy, how effective reasoning will be over a given ontology.

Qualitative Evaluation

- **ONTOMETRIC** [123]: suggests several high level criteria, such as the number of concepts, whether n-ary relationships and/or instances are supported, etc., to measure the flexibility and expressivity of the ontology. These values are associated (manually, by the person running the analysis) to a discrete category (very low/ low/ medium/ high/ very high). Using fuzzy logic, ONTOMETRIC measures the impact of each criterion on achieving the system goal and ranks ontologies based on the results. The highest ranking ontologies are recommended by the system as the best candidates to model a given system.

- **OOPS! (OntOlogy Pitfall Scanner)** [145]: is an online ontology evaluation tool that maintains a library of "pitfalls" (e.g., creation of `is_a` properties rather than using `rdfs:subClassOf`,

creating classes for what should be asserted as synonyms, presence of "orphans", namely concepts with no links) in the back end. Ontologies can be either pasted into OOPS! or presented to the tool via their URI and the automatically identified pitfalls will be shown in the interface.

- **Human-led evaluation** [136, 143]: is still a popular way of assessing ontology performance. Some guidance exists with respect to the aspects of the ontology that should be evaluated [133], namely intelligibility, fidelity (accuracy of domain representation, termed "coverage" or "similarity" in [79]), craftsmanship (compliance with design patterns), fitness (compliance with the initial requirements of the modelling problem) and deployability (ease of integrating the ontology in the information system it supports). In this context, it would be beneficial to superimpose an autonomic manager to the semantic layer in order to simplify, if not altogether eliminate, the effort involved by manual ontology evaluation.

2.2.6 Reflection and Open Issues

The research carried out in the field of semantic technologies leaves room for a series of open issues, briefly analysed below.

Ontology learning is viewed as a problem of paramount importance, referred to as the "knowledge acquisition bottleneck" [13, 22]. It is a known problem that most ontology learning methodologies (given a broad range of relevant EU research projects analysed in [93]) are domain dependant. Thus, there is a pressing need for a knowledge extraction algorithm that scales well to legacy repositories of different sizes and formats and can be successfully applied across problem domains [158], thus challenging the supremacy of ontology learning from text [143].

Web 2.0 and the Semantic Web have complementary features: unregulated, socially invested Web 2.0 annotations encourage users to contribute to folksonomies whilst also promoting repetition and ambiguity, whereas the Semantic Web requires a more rigorous formalism yet offers a well curated common vocabulary in the form of ontologies. A solution that combines the strengths of both approaches would employ well motivated users [77] to responsibly maintain an ontology, thus redefining the role of users altogether. As noted in [45], the end-beneficiary of the Semantic Web is no longer either a producer or a consumer of online content, but a "prosumer", a new identity that has to be catered for in new generation semantic applications.

Ease of use - querying on the Semantic web must allow the user to benefit from increased expressiveness while hiding the inherent complexity via user-friendly interfaces [22, 45, 121]. The limited availability of such interfaces is one of the causes for the technology transfer problem [45, 93], namely the reluctance of industry to convert research outputs into marketable products. Thus, besides placing focus on technology problems, there is a need to also address business needs by providing captivating, clean interfaces to engage and motivate users.

Decentralised and dynamic content management implies that semantic data should be (unpredictably) created, modified and discarded by various authors, whilst the semantic infrastructure adapts to the changes without compromising the consistency of the underlying ontology [22]. It is not feasible to depend on domain experts to curate the ever changing repository of semantic knowledge given that the maintenance effort would be too high and the result would fail to reflect the views of the entire community [121]. A suitable approach to address these issues, especially when modelling quickly

evolving domains [77], would be to combine expert input (for example, to build the initial version of the ontology) with community edits (in order to maintain it in the face of change).

Formal support should be developed to describe ontologies and reasoning algorithms as well as standardise semantic technologies [45].

Semantic applications should migrate from research projects and small scale commercial endeavours taken on by early adopters into open domains [45], where the advantages of semantic technologies can make a significant impact. In order to cope with the competitiveness of open application domains, ontology engineering should be carried out in a methodological, expert-overseen, collaborative way that steers away from Web 2.0 [158].

Evaluation metrics need to be improved and extended to allow comparing semantic technologies deployed to solve different problems [158]. It is also noted [77] that no proper analysis is done (at least not consistently across a wide range of applications employing ontologies) to determine whether the effort involved in creating the ontology is worth the benefits it brings. In the view of application-oriented specialists, interviewed in [141], that effort mostly consists in surmounting obstacles such as the increased complexity of semantic technologies, the insufficient quality of the available supporting software and the scarcity of success stories.

Reasoner scalability to big data is seen as a principal challenge in the field of web semantics [40, 93], as the algorithmic principles behind reasoners need to be adapted in order to cope with ever increasing volumes of data in realtime. Incomplete reasoning is proposed as a solution, as not all new data is relevant to a given query.

2.3 Hybrid Approaches

Hybrid systems implement the basic autonomic control loop with an ontology for knowledge storage and management. The rationale behind hybrids, as well as the most popular application domains they are deployed in, are presented in this section, with a few considerations about evaluation at the end. Unlike semantic and autonomic work considered separately, research into hybrids has materialised into a relatively limited number of contributions. That fact, combined with the practical, application-specific nature of hybrid research, has prevented the extraction of general trends, common technologies or underpinning methodologies as was the case in the previous two sections. More in depth critical analysis will become possible after investigating the specific contributions, thus, the reflection and open issues section was relocated at the end of the state of the art chapter (3.3.1).

2.3.1 The Need for Hybrids

The "causal relationship between the way a system represents knowledge and its level of intelligence" [13] calls for the development of powerful knowledge models that can capture contextual information from the domains they represent and reason on it, thus further developing it. If we think of autonomic systems as problem solving engines, then semantics is the fuel they should run on.

In terms of the fit for purpose formalism that the autonomic manager's knowledge should be represented in, the intuitive choice is one comprising components, connectors and constraints [88]. This is directly compatible with the way ontologies store knowledge, namely with concepts, properties and restrictions (axioms). It is also advocated [88] that a flexible and expressive knowledge format should

facilitate updating information and reasoning on it. Specifically, setting up the knowledge repository as an architectural model (that can be reasoned on) of the autonomic system would permit the verification of the managed resource's consistency after applying changes and before they actually go live (are deployed on the physical system). Injection of semantics in autonomic systems is explicitly recommended [103] to allow reasoning about autonomic features and requirements. A case is made in the same paper for mapping the connections between system components to allow for better problem diagnosis.

2.3.2 Applications and Evaluation

The vast majority of hybrid systems target the smart home (or Internet of Things) application domain [7, 70, 149, 55]. The emphasis is on coordinating pervasive devices by matching their semantic descriptions (including the services they provide) against the current active requests in the environment. There are also notable efforts geared towards anomaly detection (be it in sensor data [56] or in the wider managed resource operation [56]) and semantically enabled self-configuration of autonomic systems [38, 9].

Evaluation predominantly consists in deploying the hybrids to solve specific problems (such as mobile robot training [9], server pool size and content rendering mode management for a news website [39], smart metering [7, 56]). The way environment information is extracted and reasoned upon is explained step by step, yet, quantitative performance analysis is usually missing. There are two exceptions: FRAMESELF [7] measures the scalability of a smart metering system, whilst Construct [56] employs an aggregate function that measures sensor data precision, decay and confidence (this leads to the development of a trust measure for input readings collected by autonomic systems during monitoring).

State of the Art

The main contributions in the fields of autonomic computing, semantic technologies and hybrid approaches are discussed in this chapter. Relevant autonomic systems, components and methodologies described in the reviewed literature are grouped by the stage of the MAKE-K loop they (mostly) address. On the other hand, semantic contributions are classified with respect to the stage of the ontology engineering process that they fit best. Some critical analysis of the state of the art is also provided, by means of comparing the main features of the reviewed work and highlighting both strengths and open issues.

Running example. To better illustrate the main features of the existing approaches as well identify areas of improvement, a smart environment application scenario will be used throughout this chapter. The same example will be analysed in chapter 4 as well, to highlight the ways in which implementing the proposed Knowledge-centric Autonomic System architecture would address the issues existing within the state of the art. In brief, the smart environment to be used as a running example collects sensor information (temperature, pressure, etc.) from various monitored objects such as household appliances. Given the data available at a given moment in time, an autonomic decision maker, regulated by a set of high-level policies (or rules) and informed by a knowledge base describing the environment and its possible states, suggests an (a sequence of) action(s) to drive the managed system towards the prescribed goal state (e.g., where overall power consumption is under a given threshold). The details of this architecture and its behaviour will be gradually added throughout the **Summary and running example** paragraphs at the end of each section in this chapter.

3.1 Autonomic Systems

Given the wide range of autonomic computing applications (see 2.1.3), suggesting a clear cut classification is challenging. Some attempts to group contributions in the field consider the degree of autonomicity (see the introduction for 2.1) [88], discriminate based on system architecture (flat and hierarchical) [131], or suggest a separation based on unrelated criteria, resulting in categories such as biologically inspired, large scale distributed, component based, technique focused, etc. [104].

In the following, we suggest separating and analysing the implementation of each MAPE-K loop component from the set of applications covered in this literature review. This is particularly difficult since the stages of the MAPE-K loop are usually tightly knit [155] (most likely one of the reasons why this classification approach has not been taken before). Organising autonomic contributions based on the monitor, analyse, plan and execute stages is primarily justified by the fact that MAPE-K is one of the few pre-set, well defined autonomic structures with a wide acceptance by the relevant academic

as well as industrial communities (the other discrimination criteria used in the literature to classify/ compare autonomic applications are subject to their authors' personal bias with respect to importance and completeness). Also, this approach will highlight good practices, challenges and shortcomings with respect to specific MAPE-K stages as well as the loop as a whole, thus setting the scene for the hybrid architecture proposed in this work and the advantages it brings.

3.1.1 Monitor

All autonomic systems implement some form of monitoring that collects data from the environment and from the managed resource via sensors. The types (sources) of input data as well as the purpose it serves are illustrated in the following, on several applications.

- **Reports.** Tactical data from ground and aerial sensors as well as soldiers from the field are compiled in early autonomic systems [88].

- **Logs.** Data is collected from system logs to inform the analysis phase [80, 189] or do diagnose an operation error [41]. The systems that provide the log data range from instant messaging to geographic information applications [95].

- **Personal computers.** The autonomic desktop manager [108] monitors operating system log data in order to identify hanging processes and kill them.

- **Web services.** Data produced by web services running over a distributed network is read in and processed for load balancing [182]. Also, resident services operating in smart home environments feed their requests into Virgil [94] in order to be granted access to a specific area. AutoI [21] and 4WARD [188] are two other examples of applications retrieving data from web resources and online services.

- **Distributed, large-scale systems.** IBM's Oceano [62] collects server load data over distributed networks. CogNet [148] receives communication systems parameters (such as operating frequencies, interference magnitude, etc.). Load information with respect to medical services distributed on virtual machines in the cloud is also read in [4]. Information from databases storing system usage data is tracked by IBM's SMART [119] and Microsoft's AutoAdmin [3].

In terms of actual operation, most autonomic system sensors simply pass the collected data onwards to the subsequent stages of the MAPE-K loop. One exception is the autonomic monitor suggested in [2], that adjusts its data collection frequency to improve overall system performance.

Summary and running example. Most monitored data is retrieved from distributed networks (be it of web services or virtual servers running medical applications) via software sensors (some API support is required to read server load information and application runtime logs). There is limited scope for physical sensors, especially in smart home environments, where pervasive devices directly monitor physical inputs, such as temperature or humidity. Very little data filtering is performed, one exception being [2], that leads to the conclusion that most of the load of sifting through input data will be passed on to the analysis phase. Thus, considering specialised sensors capable of reading a limited type of inputs would help simplify the analysis module. Plan selection would also be reduced to identifying the type of sensor that provided the latest batch of data.

In the context of the running example, providing specialised interfaces to handle each physical sensor in the environment is not a scalable approach. Specifically, if the smart environment is a power plant instead of a home, new monitoring interfaces would need to be added to interpret input from industrial sensors that would not exist in private residences. This outlines a problem with exiting monitoring solutions, namely that they do not usually transfer well from one problem domain to the next.

3.1.2 Analyse

Analysis consists in matching monitored log data against a database of symptoms (namely snapshots of parameter values) associated to known problems occurring during system operation [41]. The strategy employed in the past to treat the identified symptom, if available, is re-applied or improved. This general description has been customised to fit specific frameworks or application environments, resulting in a series of bespoke implementations of the analysis module. Some of the most representative ones are briefly explained below.

- **Formal data representation.** Unstructured log data is parsed and translated into an event based format [189] using IBM's Generic Log Adapter (a tool available as part of the Autonomic Computing Toolkit). This makes monitored data easier to compare against known patterns from previous system experience in order to identify the most appropriate action plan.

- **Utility assignment.** An utility value is assigned to each service competing for dynamically allocated resources inside a grid architecture [182]. The utility is continuously recalculated in response to the increase/decrease of demand for a given service. This informs the resource allocation plan.

- **Discretisation.** A reduction function is used to turn continuous domains of values for monitored parameters into discrete sets (e.g., low, medium, high) [39]. The thresholds separating the sets are determined based on heuristics. A second analysis stage is interleaved with planning: after a section (tactic) of a plan is executed, the state of the system is explored to decide which tactic to apply next.

- **Delay modelling.** The speed of the connections between different distributed network components is estimated by collecting response delay data from probes reacting to a test signal [188]. The delays are modelled as a Gamma distributed set and compensated by automatically tuning the remaining network parameters.

- **Probabilistic parameter tuning.** The values of monitored network parameters are compared against known performance indicators for several network configurations [148]. The value that best meets the performance requirements is used as a seed to randomly generate the new value of the parameter in question. A related approach is employed by Oceano [12], where monitored events are correlated via statistical techniques in order to produce a model for input data where patterns can be identified more easily.

- **Control theory.** A prediction model is used to forecast all possible system states and pass the optimum one as a reference to a controller [147]. The former implements a feedback reaction that would bring the actual state of the system to the reference one.

Summary and running example. Given the tight connection between analysis and planning, it is sometimes difficult to extract the analysis logic from systems that fully implement the autonomic loop [95, 108]. As can be seen from the examples above, irrespective of the underlying implementation, analysis informs every step of the planning process (selecting plan steps, deploying plans on a temporary model of the system, investigating the effects and deciding whether to proceed with plan construction or backtrack to a previous version), an interaction that has a significant impact on the performance of the overall system. Thus, it would be useful to propose a methodology that captures the collaboration between the two phases of the MAPE-K loop, with the added benefit of tractability across application domains.

Referring to the concrete running example, the analysis phase would typically consist in running a query to determine, for instance, if the overall energy consumption is under a given threshold or if the pressure sensors in the fridge door indicate that the milk bottle is running low. Whereas, in the latter case, the ensuing action is straightforward (put "buy milk" in the home owner's calendar), the former situation requires more complex planning. Simply switching all electrical appliances to a low power operation mode is unrealistic, as that might conflict with other high level goals, such as keeping the room temperature above a given level. That prompts the autonomic manager to loop back to the analysis phase after implementing (or, better yet, simulating) each step of a candidate plan to make sure that none of the active policies is broken. Providing a computational framework to support this tight cooperation between the analyse and plan stages of the MAPE loop is an area of research in need of more investigation.

3.1.3 Plan and Execute

As described by IBM [90], planning represents the autonomic control loop stage when actions (selected from a set made available by the domain expert or extracted directly from the policy document) are dynamically combined to form an executable sequence (plan). The purpose of carrying out the sequence of actions via the system effectors is to get closer to the goal state. The literature describes various implementations for the planning logic suggested in IBM's blueprint, some of which are described in the following (the way plans are stored in each contribution is highlighted in bold).

- **Java beans.** CHAMPS [165] creates plans from user defined Java methods implementing legal actions. The plans are used for installing middleware (e.g., Java2EE, Tomcat, MySQL) necessary to run an online book store based on host system and middleware vendor specifications.

- **Trees.** Plans (strategies) are stored as tress [39], where branches (tactics) are dynamically selected based on analysing the current system state and its closeness to the objective.

- **PDDL (Planning Domain Definition Language).** The outcome of past actions (user defined PDDL routines) is used to advise the selection of new ones to append to the current plan [147].

Execution follows immediately after planning and consists in deploying plan actions on the managed resource or on the autonomic manager itself (usually by modifying the knowledge base). Some of the realisations of the execute stage are presented in the following.

- **Course correction.** Trajectory management plans are executed to allow autonomous vehicles to avoid obstacles while exploring uncharted space [88].

- **Configuration fixes.** Solutions produced during planning are implemented to address configuration issues for large scale enterprise systems [80] or for back-end databases supporting online services [41].

- **Load balancing.** In Kinesthetics eXtreme [95], the execution phase implies marking emails as spam or accepting/ rejecting tasks in order to meet a load quota for instant messaging and online services employed by geographical information systems. Actions having to do with maintaining the desired load balance are executed in Unity [182] as well. Oceano [62] controls server load by executing server-to-task allocation plans in distributed applications.

- **Parameter configuration.** In 4WARD [188] and CogNet [148], execution consists in setting new values for system parameters.

- **Online application maintenance and desktop management.** CHAMPS [165] executes plans about the installation of middleware for an online book store. The actions carried out by the personal autonomic desktop manager [108] consist in terminating hanging processes on a PC.

Summary and running example. The most popular trend is to delegate planning to policy authors (ECA policies directly encode plans), as this is computationally cheaper than employing a dedicated planning algorithm - the contributions taking this approach were not discussed here. Even applications that provide bespoke planning logic rarely describe it in detail [148, 39, 27]. However, it is possible to sketch a general planning process largely followed by most contributions: plan actions are selected (from a set of candidates defined by users in various formats or imported from existing libraries), combined to form a sequence (given a set of user prescribed rules and/or policies) and tested on a temporary copy of the domain model (this last task belongs to the analysis phase). This logic inspired the relevant segment of the methodology presented in 4.4.

Relative to the running example, should the energy consumption in the smart home exceed a prescribed threshold, the plan meant to drive the managed system back to the goal state would be compiled from previous successful actions taken in such situations and stored in the autonomic manager's knowledge base. For instance, if, in the past, powering down the TV, stereo and dimming the porch lights while keeping the radiators and fridge on a high setting managed to bring down the energy output without compromising ambient temperature and food quality, the same actions would be considered again for inclusion in the current plan - a decision that would either be validated or not by re-running the analysis algorithm.

3.1.4 Knowledge

The efficiency of the autonomic control loop relies heavily on the accuracy of the available knowledge. Either provided by the domain expert (state models of the managed resource, policies, rules), extracted from the controlled system (sensor logs) or produced by the autonomic manager itself (symptom databases, plan templates), the knowledge repository informs every step of the analysis and planning

stages and should therefore be stored in a format that facilitates access and maintenance. There are numerous such formats proposed in the literature.

- **Collection of symptoms.** Known issues (previously encountered and documented) of large enterprise applications form a knowledge base of symptoms [80] to be consulted during analysis. The idea is extended in [41] where the symptoms database also stores common causes for the issues on record as well as previously applied solutions.

- **Plan elements.** In the context of dynamic plan selection, knowledge represents a collection of tactics (building blocks for plans) to reuse when formulating strategies [39].

- **Spanning trees.** In 4WARD [188], knowledge consists in a mathematical model (spanning tree) supporting the computation of threshold levels for performance indicators during analysis. A hierarchical prediction model for forecasting possible system states is used in [147].

- **Parameter configuration data.** The knowledge base of CogNet [148] stores the most commonly used value along with the acceptable range for system parameters. These are updated after the analysis phase completes.

- **Classes in an OOP language.** Policy & Profile [36] is an object oriented platform specifically designed to store knowledge for autonomic applications. Classes are built to model *policy concepts* (`Event`, `Condition`, `Action`), *profiles* for managed resource components (e.g., `Account`, `Service`) and *roles*, namely abstact classes capturing facets of profiles in specific contexts (the `GP` role represents the `Account` profile in a medical context, granting exclusive access to patient data). During analysis, policy related classes are used to determine which role is applicable to a given profile under specific circumstances.

Summary and running example. All autonomic applications circulate data between MAPE components. Some systems do not organise or permanently store knowledge in any way (data is passed from the sensors to the interested MAPE component and then discarded) [95, 165, 182]. In other cases [108, 27], the way knowledge is managed is not disclosed, although, from the platform description, it can be deduced that some layer of knowledge organisation exists. As the examples above show, the format knowledge is stored in (mathematical models, databases, classes in a programming language) as well as what it refers to (symptom - solution pairs, managed resource state, plan building blocks) varies from one problem domain to the other. This heterogeneity makes it difficult to assess the impact of knowledge base contents and representation on the efficiency of planning and analysis. Another conclusion drawn from analysing the contributions above is that knowledge is a shared resource within the autonomic system's economy (all MAPE components consult and modify it concurrently, increasing the risk of data inconsistency). To address these issues, proposing a common yet flexible format for storing knowledge (such as an ontology) would be beneficial.

To provide a practical argument in favour of using ontologies, let us refer to the running example, namely the way that information about the smart environment is modelled, retrieved and maintained in/from the knowledge base:

- Modelling. Representing electrical appliances as concepts in an ontology (along with the appropriate properties) would allow the reasoner to automatically infer useful implicit

relationships, such as the fact that a fridge is a cooling device. By contrast, in an OOP model, class `Fridge` would have to be explicitly defined as a subclass of `CoolingDevice`, whereas classic relational databases do not provide any native support at all for modelling hierarchies.

- Retrieval. Let us assume that all cooling devices in the kitchen have a maximum capacity of 150L. If a query is run on the underlying ontology to retrieve the list of all devices with a capacity of under 200L, the reasoner would automatically include the fridge in the results list, even if the corresponding concept has no capacity property attached. This is because, as seen above, ontology term `Fridge` is subsumed under `CoolingDevice`. A database would not accept the `Fridge` record without a specific capacity (a consequence of the CWA), whilst, in an OOP class hierarchy, a bespoke search algorithm to find all objects meeting the requirement in the query would have to be written by the application designer.

- Maintenance. In a scenario where acceptable cooling devices have an A+ energy rating, the home owner orders a freezer with an A rating. If the knowledge base is stored in an ontology, the reasoner will subsume `Freezer` under `CoolingDevice`, causing the former concept to have two fillers for the `hasEnergyRating` property: A+, inherited from the parent, and A, prescribed by the manufacturer. This would cause the ontology to become inconsistent, prompting the system to warn the owner against completing the freezer purchase. In an OOP model, the A rating would simply override the A+ one in the superclass, causing the inconsistency to go unnoticed. The same would happen in a relational database, where the record inserted to model the freezer and the one representing a generic cooling device would simply co-exist with conflicting values (an explicit restriction would have to be put in place to prevent that).

3.1.5 Policy

The high level knowledge meant to guide the operation of the autonomic loop is usually prescribed by the domain expert in the form of policies. The various forms of policy languages as well as the main issues around policy definition and deconfliction were discussed in 2.1.2. The following provides more insight into the use of policies to regulate autonomic systems' behaviour.

- **Policies as the managed resource.** OWL-POLAR, introduced in [155] and extended in [11], is described as a policy *language*. However, considering the case studies described in [155], it is more appropriate to view OWL-POLAR as a *platform* for reasoning on policies, allowing the development of a fully fledged (meta) autonomic system where policies are the managed resource. The goal of this system is to eliminate policy redundancies and conflicts. The Generic Policy Analyser [87] is a similar (meta) autonomic system based on policy decomposition into atomic formulae. Those are matched against a bank of benchmark formulae to detect and eliminate policy conflicts. The same autonomic take on policy deconfliction is applied for systems roaming over several domains (e.g., mobile phones operating `@home` and `@work`) [154] as well as for heterogeneous networks [27].

- **Smart environment control.** Ponder2 [176] applies autonomic policy management to pervasive environments. Virgil [94] is a security system controlling access requests to smart spaces, based on policy language Rei. Deconfliction is performed either by prioritising policies or setting up

meta-policies (indicating which of several conflicting policies is to be considered in a specific context). The DRAMA system [37] introduces hierarchical policies, where each level corresponds to a military rank.

Summary and running example. A significant amount of effort has been invested in defining and managing policies (formulating languages with a complex enough syntax to represent sophisticated policies, building strategies, mostly involving semantic reasoning, to eliminate conflicts and redundancies, etc.). Given the reviewed body of literature, there seems to be very little focus on how policies help guide the autonomic manager with controlling a managed resource. Specifically, the connection between policies and the manager's knowledge base as well as the impact of policies on the analysis and planning stages[1] requires more exploration.

To illustrate this on the running example, let us explore a few ways to represent the main non-functional requirements of the autonomic system managing the smart environment, namely:

- keeping overall energy consumption under a prescribed threshold and

- maintaining optimal temperature levels in rooms as well as inside food storage devices.

The first option is to represent the two requirements as policies in a dedicated language such as OWL-POLAR. This would offer support for capturing and managing dependencies (in our case, the policies are conflicting, as reducing energy consumption implies powering down appliances, however, radiators and coolers implementing the second requirement are usually heavy consumers). The downside of such an approach is the introduction of a secondary ontology to model policy related concepts (such as `Action` and `Precondition` - see [155]) and allow reasoning for the purpose of policy deconfliction. Should the smart environment be a power plant rather than a home, where too high an energy output would cause a critical event, the computational overhead may become unacceptable. The second alternative implies defining the two requrements as rules in SWRL, resulting in faster reasoning than in the previous case, but requiring a SWRL expert to formulate the rules in the first place. Finally, the requirements may be kept outside the ontology altogether and implemented in an OOP language instead. A simple deconfliction mechanism, such as setting a higher priority for the second non-functional requirement, would efficiently avoid the reasoning overhead entailed by heavyweight solutions such as OWL-POLAR.

3.1.6 Middleware

There are several frameworks (methodologies) available in the literature for implementing the autonomic control loop. They can be viewed as generic architectures, accompanied by a set of tools to support their practical realisation, meant to guide (and sometimes automate) the autonomic manager's construction process.

- **IBM's ACT (Autonomic Computing Toolkit)** [91] is a collection of tools designed to help developers build autonomic managers from scratch. The core framework elements are: the *Autonomic Management Engine*, a JavaScript implementation for the four MAPE components sharing knowledge stored in the IBM Cloudscape database, the *Generic Log Adapter* that parses

---

[1]Beyond the trivial case of ECA policies, where no separate planning logic is needed.

unstructured log data and translates it to a standard *Common Base Event* format and the *Log and Trace Analyser* that analyses translated messages and builds a correlated view. ACT has been implemented for network services configuration [128], resource allocation in public health supply chains [14] and large system diagnosis [80, 41] (mostly exploiting the Generic Log Adapter and its parallelised version suggested in [189]).

- **IBM's ABLE (Agent Building and Learning Environment)** [24] provides autonomic managers implemented by agents that are specialised in a specific set of tasks. After appropriate configuration, the agents may be deployed to work towards a specific goal in a legacy application.

- **Ponder2, KAoS and Rei** [174] can be seen as middleware platforms as they provide a formal policy language derived from the ECA format along with a set of algorithms to run in order to build an autonomic policy reasoning engine.

- **Autonomia** [57] provides a set of bespoke components, each fitted with its own autonomic manager, for integration in distributed autonomic systems. **Accord** [117] caters to a similar purpose by providing configurable components for autonomously managing applications in a grid computing environment.

- **Web services** are not clear-cut autonomic frameworks as the other items in this list, however they are fit for purpose with respect to implementing MAPE-K components. Given that analysis and planning are versatile procedures (with different implementations and behaviours, depending on the problem domain) that need to function in realtime and, often, in collaborative environments, web services are good candidates to provide them with a practical realisation. This potential has been noted [90] and explored [184] in the research community.

Summary and running example. Most middleware solutions offer some degree of flexibility (as demonstrated by the wide acceptance of ACT to build autonomic systems for a variety of application domains), however, they are difficult to configure for custom components of the autonomic control loop (i.e., there is no direct way to integrate ontologies as knowledge storage platforms in the ACT architecture). Off the shelf solutions can be heavyweight (translating monitored data into the Common Base Event format to ensure compatibility with ACT's engine is more resource consuming than directly processing lightweight semantics or interpreting system logs in standard format). It is also worth noticing that not all middleware is a success story (ABLE has had no noticeable uptake in the relevant community).

Providing a practical implementation for the smart environment in the running example by instantiating a template architecture such as ACT would most likely introduce a significant design overhead without the added benefits of storing the knowledge base in an ontology.

3.1.7 Reflection

The reviewed applications are included in Table 3.1 to support an in-depth comparative analysis carried out in the remainder of this section. The criteria considered for each contribution are:

- **MAPE** separates the applications that provide a full implementation of the autonomic control loop from the ones that target a specific component, be it monitor (M), analyse (A) or plan (P).

Table 3.1: Autonomic applications

MAPE	Level	Application domain	K	Formal	Middleware
full	2	space exploration [88]	none	no	none
	3	self-configuring network services [128]	none	no	ACT
	3	public health supply chains [14]	none	no	ACT
	3	system diagnosis [80, 41, 189]	DB	no	ACT
	3	component repair [95]	none	no	none
	3	desktop management [108]	none	no	none
	3	policy reasoning [178]	ontology	DAML	none
	3	distributed systems management (medical) [4]	none	no	none
	4	distributed systems management [1]	none	no	Globus
	4	distributed systems management [12]	none	statistical correlation	none
	4	network management (military) [37]	policies	no	none
	4	network management [27]	none	no	Minmex
	4	network management (online) [188]	none	spanning trees Gamma distrib	none
	4	data centre resource allocation [182]	none	utility theory	none
	4	middleware installation [165]	none	no	ABLE
	4	communication systems optimisation [148]	none	no	none
	4	policy reasoning [155, 87, 154]	ontology	DL OWL-POLAR	Protege OWL API
	4	policy reasoning [94]	ontology	Prolog/Rei	none
	4	policy reasoning [176]	policies	PonderTalk	javac
M	2	autonomic monitoring [2]	none	no	none
A	3	knowledge modelling [36]	classes	OOP	none
	3	system diagnosis [6]	none	recursive NN	none
P	3	resource management [147]	none	state forecast control theory	none
A, P	4	architecture-based self-adaptation [39]	strategies	utility theory	none

- **Level** represents the degree of autonomicity on the IBM scale [88] (explained in 2.1).

- **Application domain** states the field the system was deployed in.

- **K (Knowledge)** shows the type of model (if any) used to store information for the benefit of the analysis and planning stages.

- **Formal** refers to any structured language or mathematical framework utilised to represent the knowledge model.

- **Middleware** reveals whether an autonomic development framework, API or some other third-party software was used to build the application.

The overwhelming majority of applications provide an implementation for the **full MAPE-K loop**. This shows an inclination towards using autonomic technology as a means to solve a specific task, rather

than investing in perfecting specific components involved in the control loop and making them available for use across problem domains (out of 28 applications included in the comparison, only two focus on analysis and two on planning). The study of the autonomic *infrastructure* (namely, investigating MAPE-K components in isolation as well as their interaction) is an attention-worthy gap in the field.

Most applications are situated in the **middle of the IBM autonomicity scale** (levels 3 - predictive and 4 - adaptive), which shows a competitive maturation speed for autonomic technology. To set a reference line, we can consider two other major technologies that have had a significant impact on the history of IT, namely the internet (that has taken around 20 years to evolve from ARPANET[2] to the WWW[3]) and mobile operating systems (with a roughly 15 year timeline between the release of the first smart phone[4] and the introduction of iOS[5] and Android[6]). In comparison, autonomic computing is 15 years old (considering IBM's 2001 manifesto as a starting point) and is already approaching the final milestone on the path towards achieving its full potential (the fifth and final level of autonomicity).

The most **popular application domains** seem to be system diagnosis/configuration and load balancing over distributed/grid architectures (each targeted by 9 contributions from the analysed set). These are "traditional" areas of interest for autonomic technology deployment and have been consistently targeted by the research community over the years. There is little coverage of fields such as desktop application management or large scale knowledge graph maintenance and visualisation (1 contribution for each). These are also the two application domains considered in chapters 5 and 6.

Less that half (12 out of 28) of the analysed applications use some form of resident storage for the autonomic manager's **knowledge**. Out of these, 7 employ some structured syntax (logic or programming language) to describe the knowledge whereas 1 uses a mathematical formalism (namely, utility theory). What is also striking is the fragmentation in the set of approaches to knowledge modelling: there are only 3 applications [155, 87, 154] in the entire table that employ the same ontology language (OWL-POLAR). The way that knowledge representation and management influences autonomic behaviour is thus difficult to analyse.

In terms of **middleware**, ACT has the most pronounced uptake, however, it is popular chiefly with applications targeting the distributed network management domain[7]. Besides standard tools (editors - Protege, compilers - javac and java libraries - OWL API), the use of which is inherent to the type of knowledge model employed, autonomic application reliance on middleware is limited (16 contributions do not use it at all). This raises some concerns around the flexibility and extensibility of existing frameworks (how well they adapt to different problem domains) and also the reusability of individual autonomic components (linking back to the first point of this discussion).

To conclude, it is worth mentioning that other reviews [88, 104, 131] provide an excellent coverage of autonomic applications, however, they focus on the in-depth description of each contribution (up to

---

[2]The first message was sent over ARPANET between the University of California Los Angeles and the Stanford Research Institute in 1969.

[3]The World Wide Web crystallised in 1990, with the completion of the first web browser and the first web server by Tim Berners-Lee.

[4]IBM Simon in 1994

[5]2007

[6]2008

[7]This study has not found evidence of ATC use in other problem domains, but that is not to say that this framework is not extensible.

and including the implementation level). That is only partially relevant, especially since one of the gaps in the area, identified by the previously mentioned surveys themselves, is the lack of a *common framework* for evaluation, design principles, and configuration of autonomic frameworks. Also, flexible, extensible tools that would successfully play te role of autonomic building blocks get little to no attention. These topics would be better addressed by running a high level comparisons aimed at extracting trends, common pitfalls and good practices. This would allow for the main challenges in the field to be accurately formulated such that the appropriate research directions could be suggested. This is what the presented analysis attempted to achieve.

3.2 Semantic Technologies

This section focuses on contributions in the field of ontology engineering. However, other semantic technologies, such as reasoning and querying, as well as representative applications built on top of a knowledge repository stored in the form of an ontology are also discussed. The reviewed work is classified with respect to the addressed OE phase, in order to extract good practices and eventually compile a set of guidelines on how to design, query and maintain an ontology. The categories considered here also match the four types of business value brought by semantic technologies [93]: (1) semantic metadata discovery and acquisition (maps to ontology learning); (2) meaning representation and integration (maps to ontology design); (3) reasoning, interpretation, inference and query answering; (4) presenting, communicating and acting upon knowledge (maps partially to querying, specifically results visualisation, and to decision support).

3.2.1 Ontology Extraction

The main methods utilised by semantic applications for ontology content extraction, along with the source knowledge repositories, are presented in the following.

- Automated ontology learning **from text** employs statistical tools such as word frequency measurement to extract ontology concepts and co-occurrence (two words appearing together) analysis to define properties. Applications using this approach to extract ontologies range from dedicated tools, such as Text2Onto [42], to broad spectrum ontology engineering platforms, such as GATE [46], equipped with annotation tools and natural language processing modules.

- CS Aktive Space [156] extracts data about UK computer science research **from databases, webpages and other ontologies**. The application allows ontology exploration through an interactive interface.

- Garlik [13] extracts financial data related to a given person **from various web-available sources** (credit reports, employment history, etc.) and allows the management of that data to prevent identity theft and financial fraud.

- Ontologies are extracted **from Wikipedia pages** [78]. Wikipedia URIs are used to prefix ontology elements for disambiguation.

- SPY [71] is a method for extracting software component specifications **from data generated during runtime**.

Summary and running example. As the above examples show, methods have been perfected to extract ontologies from a variety of sources. When the original data is stored in an unstructured way (free text, webpages), the ontology learning logic relies on standardised, data mining techniques meant to identify relevant concepts and the semantic links between them. On the other hand, if the legacy data is well structured (databases, other ontologies, system logs), the computational cost of statistical analysis is no longer justified. Instead, bespoke OL algorithms are developed to exploit the format of the source repository and extract concepts/relationships significantly faster. Of course, this gain in efficiency is obtained at the cost of portability, as the OL algorithm is specialised to interpret one specific format only. Addressing this separation would require an extraction technique that is both computationally efficient (exploits the structure of the source data) and flexible (is capable of parsing a variety of formats). A template for such an algorithm is proposed in 4.3.2 (parsable formats are row organised, with each row describing a property with its name, filler and domain).

In what concerns the running example, a good efficiency - flexibility trade-off may be achieved by implementing the ontology extraction algorithm in a non-DL language (such as Java). Since the equipment in the smart environment is likely to be documented in a standardised format (namely, the products' description would be made available by the manufacturer in the form of spreadsheets or a similar document type), a third party tool capable of parsing that data would be relatively easy to set up, would run at a low computational cost and would transfer well to other application domains (where legacy knowledge is stored in a compatible way).

3.2.2 Semantic Reasoning

Semantic reasoning is exploited in several stages of the software engineering process [138, 139], as illustrated by the applications below. Besides the examples in this section, semantic querying, also a reasoning-related function (see 2.2.3), is widely used for web-based question answering – a few of the most important semantic search engines underpinned by this type of querying are presented in section 3.2.3.

- **Domain specific modelling.** A domain specific language, the syntax restrictions of which are stored in an ontology, is used to formulate models for telecommunications networks [183]. To aid the process, reasoning services are employed at various stages: *satisfiability verification* is carried out to check whether a candidate model complies with the syntactic restrictions held in the ontology, an *explanation for inference results* is provided to suggest contingency measures for pre-set faults (such as a missing component in the structure of a physical device) and *querying* is supported to determine whether a network model can be extended with a specific component while guaranteeing that the syntax ontology remains consistent.

- **Web design support.** A common path taken by non-expert users building their own websites is to aggregate ready made web components published by vendors such as Google, Microsoft, etc., with the expectation that their efforts will not require advanced technical knowledge. The currently available catalogues of such software tools are heterogeneous and have interfacing issues (a Yahoo-published component may not play well with one from a Google catalogue). In response, a universal repository is proposed [118], centralising components from all publishers.

The solution is underpinned by a universal model (developed in DL by translating the XML metadata accompanying component descriptions) capable to represent all web tools, regardless of their publisher. A reasoner (HermiT) is used to determine whether a given component is a subclass (*subsumption*) of a concept in the universal model or an instance (*query answering*) of it.

- **Runtime selection of software implementation variants.** The different runtime states of the software supporting a modular video platform are represented in an OWL 2 ontology [74]. When the software is running, reasoning services (namely, *subsumption*) are employed to detect its current state (low or high energy consumption). If a non-functional requirement is broken (the energy consumption is too high) the reasoner is used (via *semantic querying*) to determine which of the available implementations for the active software components would bring the system to an energy efficient state. The authors note that semantic reasoning, specifically the open world assumption it implements, is particularly useful in this case, as representing software runtime states implies handling incomplete information without sending the model in an invalid state. Another application that reports the increased performance of OWL 2 reasoners, this time when compared against graph traversal execution environments, implies executing ontology-modelled clinical practice guidelines [92]. In some applications, a reasoner is the only technology available, as is the case with debugging logically erroneous knowledge bases modelling a medical terminology about intensive care patients [151].

- **Streamlining intelligent agents communication.** Third party reasoning engines (R-DEVICE and Prova, as well as services capable of dealing with incomplete information, such as DR-DEVICE and SPINdle) are used to translate the semantics of messages exchanged between intelligent agents [111], an approach that is reportedly computationally cheaper than providing all agents with local implementations of a translation algorithm. The proposal is demonstrated on a brokering application, where one agent uses an automated real estate service to find properties that fit a set of pre-determined criteria. Another application using bespoke reasoning algorithms to manage intelligent agent systems uses the Narrative Knowledge Representation language to model the environment (e.g., objects such as a fridge and its contents) and a set of rules [18]. Based on this model, a reasoner suggests actions for the assisted person, for instance, buying more food.

- **Pervasive environment service personalisation.** User preferences and needs are represented in an OWL ontology feeding into a personalisation service, supported by the Pellet reasoner [162]. Based on a SWRL rule base, the tool matches profile information against services available in the environment (e.g., switching the language used by a ticket machine, given the user's nationality). In another application, reasoning services are employed to match (incomplete) user profiles, this time against each other, on a dating platform [31].

- **Meta-reasoners.** A common interface, TrOWL, is proposed to support a number of different reasoners [171]. It is meant to facilitate the translation of, for example, conjunctive queries (by using semantic approximation) between OWL 2 DL and OWL 2 QL, but can support other reasoner implementations, such as FaCT++ and Pellet. The authors argue that the TrOWL infrastructure enables the reasoning over and querying of large scale OWL 2 ontologies at organisational level.

Summary and running example. Most applications use reasoning services in their standard implementation to support the desired high-end functionality. Semantic reasoning is rarely employed in conjunction with some autonomic decision making mechanism (for a few exceptions, see section 3.3) – a research gap worthy of further investigation, as autonomic elements rely on the accuracy (consistency) and accessibility (fast and easy to understand answers to queries) of their knowledge base. All these are standard reasoning services, as shown in 2.2.3.

Let us illustrate how semantic reasoning services may be used to support the autonomic decision process in the case of the running example:

- Subsumption. A shown in 3.1.4, the reasoner infers automatically that the ontology concept modelling a fridge subsumes the one representing a generic cooling device. Thus, when a new appliance is installed in the smart environment, the corresponding ontology term is inserted in the right spot of the concept hierarchy, without the direct implication of the (human) domain expert.

- Satisfiability. The typical interaction between the analysis and planning stages of the autonomic control loop implies simulating candidate plans and observing their effects before approving or further improving them. In this process, the outcome of a given step inside a candidate plan is modelled by adding the appropriate concepts to a copy of the ontology. For instance, if one of the plan's actions is to buy a more energy efficient freezer, the concept representing the new appliance will be inserted in the simulation copy of the ontology. If the home owner misguidedly chooses a freezer with an energy rating that is lower than that allowed for cooling devices, the discrepancy would prompt the reasoner to flag the simulation copy of the ontology as inconsistent. This would inform the analysis module to reject the unsuitable freezer model and consider another action for insertion in the candidate plan.

- Synonymy. Let us assume that the fridge in the smart home is replaced with a newer model that is called "refrigerator" in the accompanying technical specification. Given that all properties for the new concept are correctly set, the reasoner will infer that the newly inserted `Refrigerator` concept and the previously existing `Fridge` one are synonyms. In addition to maintaining the simplicity of the ontology's structure, synonymy detection improves querying, as both terms are now recognisable as search keywords.

- Querying. In order to reduce the overall energy consumption of the smart house, one candidate plan may prescribe switching off all appliances with a capacity grater than 150L. While this plan is being analysed, it becomes relevant to determine which devices have a capacity that exceeds the given limit. A typical reasoner would resolve that query automatically, thus supporting the MAPE loop (in)validate the plan under consideration.

- Inference explaining. Implementing the actions, such as buying a new freezer, of the plan suggested by the smart home's autonomic manager is, ultimately, the home owner's responsibility. Providing the reasoning chain (2.2.3) in support of a recommended plan would help the owner better understand the system's suggestions before carrying them out (or ignoring them).

3.2.3 Ontology Querying

Extracting relevant data on the Semantic Web provides the user with much more than a set of exact syntactic matches to the keywords in the initial query.

- **Partial matches** to the query keywords complement the exact ones. This is possible due to semantic tag synonyms, stored in the underlying ontology, that are likely to match resources an exclusively syntactic comparison could not have identified.

- **Suggestions** are also provided alongside the query results to guide the user towards new relevant resources to explore. The (semantically enabled) Knowledge Graph displayed by Google on the right hand side of the results page contains such recommendations in the `People also search for` section.

- **Visualisation** of semantic query results has the potential of being more intuitive than the syntactic alternative. Of course, some semantic search engines (for instance, Swoogle) display results in the classic list format, yet, RDF graphs lend themselves well to symbolic representations, that show both the matching ontology nodes and their links (e.g., OWLViz output in Protege).

- **Natural language processing** is a fast developing area that is enhancing query answering with the capacity of interpreting questions formulated in the user's language of choice. IBM's Watson is a remarkable breakthrough in this field.

A selection of relevant applications featuring semantic querying capabilities is presented in the following, with a special section dedicated to results visualisation.

- Corese [44] is a semantic search engine that crawls web resources provisioned with graph annotations. These are complex semantic tags containing graphs with nodes representing ontology concepts (that apply to the annotated resource) and edges describing relevant ontology properties. Semantic search is done by "projecting" the query graph against annotation graphs. The projection can either yield an exact match or an approximate one, measured in terms of "semantic closeness" via a metric called "ontological distance". Simply put, that represents the shortest of the subsumption paths between the two given ontology concepts and their nearest common super-type (a formula is provided in [44]). Corese can also measure "contextual closeness", where the two ontology concepts are semantically distant but share other common features (via properties other than inheritance).

- LarKC (Large Knowledge Collider) [17] is a distributed infrastructure that enables interleaving reasoning threads thus improving scalability. The underlying logic relies on incomplete reasoning, a type of reasoning that targets only the segment of the ontology that is relevant to the user query. LarCK relies on plug-ins, each implementing a certain aspect of the reasoning logic and equipped with their own API. It is the responsibility of the LarCK users to build a workflow from these plug-ins to fit the task they need to solve.

- WebPIE [177] is an inference engine, therefore it implements the reasoning logic supporting semantic querying. It is based on MapReduce, a programming model that pre-processes the data

to undergo logical inference, thus facilitating effective distributed reasoning (running on clusters of up to 64 machines and linearly scaling to datasets of 100 billion triples).

Query Result Visualisation

The major challenges concerning the application of semantic technologies to big data have to do with *parsing* (extracting ontologies from large volumes of legacy information), *reasoning* (performing computationally expensive logical operations, such as inference, to support query answering) and *visualisation* (exposing the data to the end user in a clean, connected and navigable view). The last of the three is of particular importance, because, regardless of the computational power, elegance and scalability of the algorithms running in the background, the advantages of employing semantic technologies cannot be experienced (not even acknowledged) in full without an intuitive interface to display their output. Hence, this section is dedicated to contributions that target the visualisation of semantic query results (for an alternate classification, see [97]).

- Visualising query output can be made more flexible by enhancing the display with a recommender [58]. Viewers can use this tool to indicate which features (zooming, focusing, incremental expansion, etc.) they consider important (with weights). A score is computed accordingly and the appropriate view of the ontology is produced. The main benefit is that users have the possibility to customise their ontology viewing experience, depending, for instance, on the size of the underlying RDF graph (smaller ones are better viewed on one zoomable screen, whereas larger ones should be incrementally explored).

- KC-Viz [130] is an ontology visualiser offering features such as high level overview, zoom capability and filtering of irrelevant details. This tool reduces ontology visualisation to the focus vs context problem, namely allowing the viewer to get relevant information displayed clearly and in full size while still being aware (having a perspective) of the entire ontology. The provided solution is termed "key concept extraction" and applies psycho-linguistic criteria to identify the most knowledge rich concepts in the ontology. Those concepts are included in the initial view that the user can afterwards customise.

- Optique VQS (Visual Query System) [164] uses an graphical interface for users to express their query in an intuitive, informal way. Specifically, the platform offers a set of graphical widgets that users have to assemble - like a visual jigsaw puzzle - to form a query. This is subsequently translated into SPARQL by a software layer and passed on to the reasoner. The graphical primitives approach to query construction is also taken in the case of VOWL2 [120] a visualiser designed specifically for OWL ontologies.

- Single feature driven, less configurable ontology visualisers are also available. The Protege [82] class explorer offers an indented list of ontology concept hierarchies, whereas the GraphViz tool, also available in Protege, displays concept and properties in a node-link (graph) view. Jambalaya [167] and CropCircles [185] provide good quality zooming (the latter strays from the classical node-link pattern and displays concept hierarchies as concentric circles). Ontology rendering in 3D is available in OntoSphere [97]. In Cytoscape [163], a tool developed for biomedical ontology

visualisation, nodes are automatically repositioned on the canvas to eliminate overlaps and ensure maximum visibility.

Summary and running example. Based on the reviewed literature, there are two main approaches to ontology visualisation: displaying the concepts and properties as an indented list or as a graph (node-link view). The former provides a clean, easy to understand overview of the class hierarchy, however, other types of relationships between concepts (that is, every ontology property, besides "is_a") are displayed in a separate indented list. In contrast, graph views include both the concepts and their links (hierarchical or of a different type) in the same view. However, in the case of sizeable ontologies, providing such a complete outlook compromises clarity, as the canvas may become overcrowded.

It is thus justified to ask which of the two visualisation approaches is better suited from the end-users' viewpoint. This has been thoroughly analysed [63] in an experiment where users were presented with an ontology in indented list format (displayed in Protege) and another in node-link view (generated with a JavaSvript tool). The participants were asked to correlate the two ontologies (map them against each other) to establish overlaps, gaps and inconsistencies. The conclusion of the study was that indented lists allowed the correlation task to be performed better (the two ontologies were mapped more accurately), whereas, with node-link views, the task was performed faster. The authors of the study state that the statistical differences between measurements taken on the group that worked faster and the one that performed better were very small, thus there is no numerical evidence to recommend one visualisation technique over the other.

The outcome of the above experiment gives rise to another question about the investment in graphical tools for ontology visualisation being worthwhile or not. The answer is most likely negative, if the purpose of the experiment is ontologies' *correlation*, a task that could be achieved without a sound overall understanding of the individual ontologies involved. However, the verdict is less straightforward when the goal is ontology *exploration* and *curation*. This new context implies providing viewers with a coherent perspective of the ontology, one that is difficult to gain from an indented list where concept connections are displayed separately. It is also worth mentioning that the graphical tools reviewed above offer no support for editing (currently this is done through editors such as Protege, which is easier than directly editing the ontology's support xml file but still requires text manipulation). Research efforts is this area seem to have been concentrated on simplifying query formulation and providing ways to configure the manner in which the ontology in displayed, rather than facilitating ontology modification through the displayed graph.

In reference to the running example, the main beneficiary of the autonomic manager's operation is the home owner. Besides those actions that can be performed automatically (e.g., powering off devices over a given capacity), certain tasks (such as purchasing more energy efficient appliances) are likely to be left at the discretion of the human user, who thus takes on (part of) the role of the execute component in the MAPE-K loop. To aid the home owner in this new capacity, it would help to expose the knowledge that the autonomic manager based its analysis and planning algorithms on, in order to suggest an actionable plan. Being able to explore the manager's underpinning onotlogy in a graphical, intuitive format would increase the user's level of trust in the autonomic decision process.

3.2.4 Reflection

To set some common ground between this analysis and the one presented in the previous section, roughly the same criteria were used to categorise semantic contributions as in Table 3.1 (the two that have been omitted are **Level**, as all reviewed semantic applications are meant for the Semantic Web rather than Web 2.0, and **K (Knowledge)**, as that is universally stored in an ontology). Thus, Table 3.2 describes the reviewed applications with respect to the items in the list below.

- **OE** separates contributions based on the phase of the ontology engineering process they (mostly) target (ontology learning, querying, reasoning and visualisation).

- **Application domain** states the field the system was deployed in.

- **Formal** refers to any structured language or mathematical/ description logic framework underpinning the contribution.

- **Middleware** reveals whether a support tool, API or some other third-party software was used to build the application.

- **Explore** indicates whether the knowledge model is exposed to the user (can be explored through an interface) or transparently processed by other application layers.

Table 3.2: Semantic applications

OE	Application domain	Formal	Middleware	Explore
OL	text parsing [42, 46]	statistical analysis data mining	none	no
	online research data parsing [156]	no	none	yes
	financial data compilation [13]	no	none	yes
	Wikipedia data extraction [78]	no	none	no
	specification generation [71]	no	none	no
query	semantic search [44]	graph comparison	none	no
	semantic search (incomplete reasoning) [17]	none	inference plug-ins	no
	semantic search [53, 159]	graph modelling	none	yes
	visualisation [58, 130, 167, 191, 97, 163]	no	graphics plug-in	yes
	visualisation [164]	SPARQL	graphics plug-in	yes
	visualisation [120, 82]	no	OWL API	yes
reasoning	distributed reasoning [177]	MapReduce	LarKC Hadoop	no

Application domains. The reviewed applications were selected based on their practical nature: they are used by either the academic or industrial community to effectively solve (an aspect of) a real problem. Thus, by analysing the data in Table 3.2, it follows that most semantically powered practical implementations target two areas: the retrieval of data from heterogeneous legacy collections and semantic search (including semantic querying and reasoning, which semantic search is ultimately

based on). This conclusion is supported by wider-scope semantic technology applicability assessments [93] where it has been found that data integration tools represent 67% of all Semantic Web applications (with semantic tag generation accounting for 24%) whilst semantic search takes up a segment of 45%. What is worrying about these statistics is that there is no significant percentage allocated to "complete" applications (to populate the topmost level of the stack in Fig. 2.1) that reuse and integrate these tools. That is not to say that top level applications are entirely absent, just that they employ in-house ontology learning, querying and visualisation tools that are built from scratch and customised to fit the requirements of a specific problem domain only.

Semantic search. Well established semantic search engines such as Swoogle display a justification for each result they produce in response to a user query. A justification is a simplified logical consequence thread that explains how each result was found, in other words, why it is relevant/ connected to the keywords in the search query. That insight is somewhat dwarfed by the fact that the results are displayed in a list, thus masking the very connectivity that the justification is meant to bring forward. On the other hand, other search engines display query results in a list as well as in the form of a graph, e.g., Google Knowledge Graph, thus aiding the user in forming a better understanding of the field by exposing the connections between topics. This shows, at a practical level, the added value that semantic technologies bring to the user's experience and calls for a similar incrementally expanding[8] visualisation system to be made available for other, more specialised fields (e.g., career related knowledge).

Ontology learning. Current automated OL techniques (first five rows in Table 3.1 are relevant examples) do not entirely eliminate human experts from the information extraction process. At the very least, human input is required in the ontology validation stage. Rather than pursuing the complete automation of OL techniques, it is worth considering delegating this task (at least partially) to humans, following the Amazon Turk[9] principle. This is not a return to the "old ways" of giving domain experts exclusive authorship of ontologies, on the contrary, it addresses the main issues related to that approach, namely effort and bias. By having an entire community extracting knowledge from web resources and storing it in the ontology, the effort that would have been required from one (or a limited number of) experts is divided. Also, the ontology will no longer reflect the view of one person (a problem known as author bias) but that of the entire community (termed "community contract" [78]). Instead of money (as is the case with Amazon Turk), the "extractors" may be motivated by the very knowledge they discover (especially if it is about a field such as the careers one, where professional assistance is usually paid for).

Ontology size. Successful ontologies, that is, with a reported positive impact on the performance of the systems they support, are small and shallow (with a limited hierarchy depth) [77]. This minimises the commitment costs, namely the effort required of the user in order to become familiarised with the ontology content. Since small, shallow ontologies cannot accurately model large knowledge repositories, in order to minimise the commitment cost, it is sensible to display one portion of the underpinning graph at a time. A visualiser that allows incremental exploration and quality zooming is critical in this context.

Evaluation. The reviewed contributions are evaluated implicitly, by testing the application they operate in (the experimental results obtained for the host system are assumed to apply to the semantic

---

[8]Clicking on an item, for instance, the photo of Rooney Mara, in the Knowledge Graph generated by a search for director Steven Soderbergh will display a new Knowledge Graph with linked data about the actress. The process can be continued indefinitely.

[9]https://www.mturk.com/mturk/welcome

components as well). Moreover, the availability of data driven (quantitative) analysis methods is restricted (see 2.2.5) whereas the qualitative evaluation options fail to reach the level of acceptance necessary to form a coherent platform for (semi-) standardised semantic technologies assessment [133, 145]. To better understand the role played by semantic tools (especially ontologies) within the complex information systems they are integral to, experimental analysis should also target the interaction between semantic elements and other architecture components (such as autonomic managers).

Unifying logic. This layer of the semantic stack is meant to provide a framework for the integration of rules and queries as well as a standard for how they are to be deployed on ontologies and taxonomies. This endeavour is still under way [140], with efforts targeting the integration of rules into the OWL DL logic [106], a formal way of running SPARQL queries over OWL ontologies [107] or a framework for developing trust in the social web [132]. These represent remarkable contributions on the unifying logic layer, that nevertheless fail to reach the necessary critical mass to be endorsed as a W3C standard.

3.3 Hybrid Approaches

The various contributions in the covered literature that feature both an autonomic and a semantic layer will be described in the following, with emphasis on the interaction between the two layers (namely, the way semantic value is exploited to enhance autonomic behaviour).

FRAMESELF

This platform [7] supports self-configuring machine-to-machine systems in the wider application domain of the Internet of Things. For example, in a smart metering application, light sensors, lamp actuators and smart phones communicate with each other to monitor energy consumption in the house. The goal of FRAMESELF is to automatically build the communication channels to support that. Semantic descriptions (in terms of both configuration and behaviour) of each of the involved machines are available for the autonomic manager to compile and use to react to change. Each module in the MAPE loop has a fixed, application independent part (inference engine) and a knowledge model specific to the problem domain. The monitor gets information, via specialised sensors from the managed machines and matches that input against an M2M ontology to detect if anything has changed in their configuration/behaviour. The detected changes are passed on to the analyser that matches the profiles of the machines where the changes originated to its own specific ontology to detect the appropriate communication channel to deploy in order to communicate the change to other system components. The planner receives that information and matches it against its ontology to extract the communication services and configuration parameters necessary to deploy the communication channel in the real M2M system. The deployment graph is passed on to the execute component that matches it against its ontology to find a description of the appropriate actuators to use in executing the actions (nodes in the deployment graph) provided by the planner. The actuators are then used to run the plans and build the appropriate communication channel.

Autonomic-semantic interaction. Every MAPE component uses a specific part of the available ontology to inform its operation (SWRL rules are employed in the process). New information is being inferred at every step, yet there is no actual danger of corrupting knowledge via concurrent access.

Construct, Systems Management Ontology

The Construct framework [56] reduces uncertainties in sensor data by modelling it in an ontology used for disambiguation. Specifically, log data is saved as RDF triples and compared across sensors: for instance, one sensor states that `Waldo isIn RoomA`, according to another, `UserAbc isLoggedOn PC1` and the ontology also contains axiom `PC1 isIn RoomA`, therefore it can be inferred that `Waldo` is the user logged on `PC1`. The same mechanism can be used to detect errors in sensor data: if a third sensor states that `PC1 isOff true`, then `Waldo` cannot be logged on and one of the sensors is malfunctioning. This supports autonomic systems where the operation of the control loop is based on discrete events monitored via separate sensors. This is done by semantically aligning event data that would otherwise be difficult to interpret and, more importantly, trust. The paper also provides some means (an aggregate function that measures sensor data precision, decay and confidence) to compute a log data trust score. This places Construct on the trust level of the semantic web stack (Fig. 2.1). IBM's Tivoli Monitoring solution employs the Systems Management Ontology [115] to map raw data collected from the managed resource against stored logical objects to detect, correct and predict anomalous behaviour.

Autonomic-semantic interaction. Ontology consistency (performed by the reasoner) is used to identify sensor malfunctions. In Tivoli Monitoring, the ontology facilitates a preliminary analysis of sensor data to detect anomalies in the operation of the managed resource.

Autonomic Semantic Desktop

This is an application [29] that aids users with the management of their personal data (e.g., updating webpage profiles, organising large photo collections, etc.) The autonomic manager monitors the web for resources that are relevant to the user (relevancy is determined by matching the metadata associated to those resources against the user's profile ontology). The data is then used to maintain the user's local information (e.g., if an event that is in the user's Outlook calendar has changed venue on the official website, the calendar appointment will be updated accordingly). The autonomic manager also detects patterns in the user's browsing activity that are later on used to customise their experience (for instance, by automatically disabling cookies if the user has manually done so, repeatedly, in the past).

Autonomic-semantic interaction. An ontology is build to store the user's profile (in a broader sense, including friends, browsing preferences, calendar appointments, etc.) by extending schemas and taxonomies available online, such as the FOAF (Friend of a Friend) ontology, the Dublin Core vocabulary, with individuals specific to the user. The ontology is constantly updated and consulted to decide whether detected web updates are relevant to the user or not.

Autonomic and Ontology Driven Architecture, Aesop

Service oriented and event driven systems represent a collaborative environment where producers and consumers engage in transactions [70]. This interaction is managed with the help of an "eco-system-wide" ontology capturing the semantic descriptions of services and events. A Semantic Bus is in place to allow the manager to read the semantic annotations (about the type, role, location, etc.) associated to each of the connected entities (in a smart home, they are washing machines, lamp, window blinders' controllers, etc.). The plans developed by the manager are communicated to actuators via the same Semantic Bus. In Aesop, the way end users engage with services is monitored via terminal-generated

reports [61]. The extracted information is compared against an ontology containing service models and the outcome of that analysis is used to plan improvements in service configuration.

Autonomic-semantic interaction. The description of a device that reported a new event in the environment is matched against the ontology and used to activate the condition part of one of the available semantic rules. The action part of the triggered rule will identify the services that are capable to manage the event. The devices that offer those services will then be notified by sending the appropriate message over the Semantic Bus.

Learning Design Support Environment

In education theory, learning design is the process of integrating elements such as taught content, learning styles and educational resources (books, diagrams, digital media) in a coherent framework. An ontology is proposed [38] that captures the generic description of these elements. At runtime, the user (learning designer) instantiates the relevant part of the ontology by creating individuals modelling the current learning context (e.g., specific learning outcomes or session formats). A "context path" is created to match the user input to other relevant parts of the ontology and suggest the most appropriate learning design.

Autonomic-semantic interaction. The generic ontology instantiated at runtime supports *self-configuration*. Specifically, the elements of the learning environment are assembled automatically, based on the context information relayed by the user.

Architecture-based Self Adaptation

This platform [39] employs an ontology derived from known system administration tasks to automatically adapt the server pool size and the content rendering mode (between graphical and textual) of a fictional news website Z.com. The goals are minimum response time to the user requests, quality of delivered content and low server provisioning expenses. The autonomic manager uses a reduction function (similar to the discretisation technique employed by the KAS instance in the SAR problem domain - 5.2.1) to control the size of the state space. When a change (e.g., an increase in response time) is detected in the environment , the system will select a response strategy (treat it as a performance issue or a security issue, such as a DoS attack) based on the experiential knowledge of human system administrators available in the ontology. A mechanism from utility theory is used to select from several strategies suitable to address a given symptom.

Autonomic-semantic interaction. The ontology serves as an intermediate layer between human system administrators and autonomic machines, translating the experience and reasoning patterns of the former into a format interpretable by the latter.

Context Inferring in the Smart Home, Autonomic Service Bus

Contextual information (e.g., the level of brightness, the temperature, etc.) is collected from a smart environment via sensors [149]. The data is stored in an ontology where it is reasoned upon in order to compile a set of actions - they will be executed to "correct" the environmental state with respect to a prescribed goal. Similarly, the Autonomic Service Bus [55] employs an ontology for storing semantic descriptions of providers and requesters of services (unlike AODA, that stores descriptions of the services

themselves). This knowledge may also be viewed as contextual, in a service oriented architecture. ASB is the only hybrid application, within the scope of this literature review, that stores policies in the ontology.

Autonomic-semantic interaction. Reasoning on the ontology knowledge facilitates the analysis of *context* in different environments (connected devices in a smart home and service oriented architectures) thus allowing the construction of effective plans (in terms that the actions they contain are targeted specifically at the current context).

OASys

This is an ontology that captures the vocabulary for the autonomic systems engineering process [9]. It contains common elements that all autonomic architectures share, thus supporting the process of instantiating them in different problem domains. Specifically, an autonomic system engineering methodology can be extracted from OASys, in an attempt to create bespoke implementations with limited input from a human software architect. The ontology is used to design an autonomic system for controlling mobile robots.

Autonomic-semantic interaction. As in the case of LDSE [38], the ontology supports *self-configuration* of autonomic systems. Specifically, the semantic descriptions of architecture elements and their connections are used to select building blocks and assemble them in working designs. It is important to understand that OASys stores knowledge about the architecture of the autonomic manager and not about the managed resource that the manager will be deployed to control.

3.3.1 Reflection

The reviewed hybrid contributions are centralised in Table 3.3 and organised with respect to:

- **Application domain:** the problem/ environment that the hybrid solution is deployed in.

- **Formal:** any mathematical, logic or structured language formalism that the hybrid solution employs.

- **Middleware:** support tools, platforms or pre-existing systems that the hybrid solutions are based on.

- **Learning:** whether the semantic layer featured by the hybrid solution supports the unsupervised inference of new knowledge.

The main points distilled from the set of hybrid contributions covered in this section are presented in the following.

Learning. The most important conclusion drawn from analysing the presented hybrid architectures is that they all support unsupervised learning. This highlights the importance of inference (the operation, performed by the reasoner, that underpins learning) in facilitating autonomic behaviour. To mention just a few examples, context inference is performed in [38, 149, 39]. Alternative knowledge representation platforms (relational or NoSQL databases, object oriented models) do not natively support learning in the described sense.

Middleware. There is a pronounced trend that consists in building hybrid systems from scratch. Apart from one, none of reviewed frameworks make use of the design suggestions or tools available in

Table 3.3: Autonomic-semantic hybrid applications

Application domain	Formal	Middleware	Learning
Internet of Things [7]	SWRL	no	yes
log data disambiguation [56]	no	no	yes
anomaly detection [115]	no	Tivoli Monitoring	yes
personal data management [29]	no	no	yes
smart environments [70, 149]	OWL-S, WSDL, SWRL	no	yes
service session configuration [61]	no	no	yes
learning design [38]	no	no	yes
response strategy selection [39]	utility theory	no	yes
service management [55]	no	no	yes
architecture selection [9]	no	no	yes
semantic web agents [170]	no	SERSE	yes

the separate autonomic and semantic research fields (at least, not the ones identified in 3.1.7 or 3.2.4). There is thus a need for more flexible, configurable "building blocks" (autonomic and semantic alike) easy to assemble in an effective hybrid design.

Evaluation. The "inner mechanics" (ontology querying, configuration of communication channels between the semantic layer and the MAPE modules, etc.) of every reviewed hybrid system is explained in great detail, with focus on specific implementation scenarios. In sharp contrast, there are very few performance measurements available (the only exception is a quantitative study of the way monitoring and analysis scale to a large number of system events detected at the same time [7]).

Summary and running example. The reported efficiency of the reviewed applications demonstrates that hybrid architectures, combining autonomic elements and semantic technologies, can be successfully implemented in realistic problem domains. Beyond this conclusion, there is little value that can be extracted from the above systems and used to design and implement a hybrid autonomic manager for the smart environment in the running example. The applications presented above do indeed use reasoning services (particularly subsumption and querying) to aid autonomic decision making at some level, yet the way these interact with the MAPE components is specific to the targeted application context. There are no generic guidelines or algorithmic building blocks (tools) applicable across problem domains that may be used to manage a smart home. Also, the evaluation carried out to measure the performance of the reviewed hybrids is only pertinent to the specific problem being addressed. Concretely, the evaluation results (qualitative and quantitative alike) presented in the papers consulted for this section offer nothing that would inform the designer of the smart environment management system about adopting a semantic-autonomic hybrid approach or opting for something else altogether. The following chapter attempts to address these gaps by proposing a general architectural framework for ontology supported autonomic elements alongside a more extensive evaluation approach, with the potential of being relevant to more than one application domain.

A General Framework for Knowledge-centric Autonomic Systems

KAS (Knowledge-centric Autonomic System) is a blueprint supporting the creation of autonomic managers where knowledge is represented and maintained within a semantic layer. The open issues and research challenges, extracted from the autonomic computing literature, that motivate the KAS architecture are presented in 4.1. The same section also analyses two main problem classes and their main characteristics that call for a knowledge-centric autonomic management solution. The following sections (4.2, 4.3 and 4.4) focus on the KAS components:

- **an architecture** featuring passive components (that are *written*) and active components (that are *executed*)

- **a set of tools** (software applications in their own right) that implement the active architecture components and generate the passive ones (or use them in some other way)

- **a methodology** describing the execution flow of the proposed tools.

Emphasis is placed on the interaction between the semantic and autonomic layers with appropriate justification for every design decision involved in the development of KAS. The overall purpose of KAS, reflected in the way components are structured and connected, is maximising the advantages of semantic technologies (chiefly, reasoning) in order to support and enhance autonomic behaviour.

4.1 Rationale

The relevant literature contains numerous references to insufficiently addressed research questions directly related to autonomic management, specifically the design of the underpinning architecture and the practical implementation of the appropriate components and algorithms. These open issues outline the need for a knowledge-centric architectural framework designed to support autonomic decision making, as discussed in the following.

- Knowledge is a key piece in the design of the autonomic element, as it informs the operation of all MAPE components [101, 88]. It is thus paramount to identify a semantics sufficiently expressive to capture the subtleties of the available knowledge, whilst structurally simple enough to allow autonomic components to interpret and reason upon it in a computationally feasible manner [5]. An ontology, along with the accompanying OWL reasoning services, makes for an attractive candidate, as it inherently supports unsupervised learning (automated logical inference) as well as satisfiability checks to maintain the knowledge base consistent under concurrent access. This is relevant to the architectural aspect of research objective O1.1. (1.4).

- The autonomic computing proposal has known a wide practical acceptance (as shown by the variety of implementations analysed in 3.1), yet most of the existing renditions are application specific, offering little insight that is transferable to other problem domains [5]. This calls for a set of open, extensible tools that can be easily integrated in the general autonomic architecture and perform its functions in a variety of practical scenarios [150, 112]. As set out in research objective O1.1., KAS encompasses several active components supporting autonomic operations (such as knowledge acquisition or analysis and planning algorithms) that feature a generic structure – that transfers well to different applications – and, a the same time, may be customised to fit the specifics of a given problem domain.

- In spite of the clear architectural blueprint of the classic autonomic element suggested by IBM [90], complex practical implementations are structurally brittle, relying on subtle co-dependencies between non-standard components that are difficult to understand and challenging to replicate [5, 103]. Consequently, it would be beneficial to propose a consistent methodology that captures such interactions in an algorithmic way and can be used to inform autonomic design across application domains. This is in line with the methodology aspect of research objective O1.1.

- One of the reasons behind lack of cross-domain applicability of autonomic implementations is the in-house nature of the evaluation techniques. In other words, the reported metrics, quantitative and qualitative alike, tend to be relevant solely to the problem the system was designed to solve – based on these alone, it is difficult to gage how the same implementation would fare in a different practical context [150]. KAS components are configured to rely on mechanisms (e.g., dynamic plan construction and reuse) that can be evaluated via metrics (such as the index of learning that measures the system's capacity of exploiting its own experience) relevant to a variety of problem domains. These evaluation considerations are captured by research objectives O2.1 and O2.2.

The problem classes analysed in section 1.3 feature a set of common requirements that provide additional (practical) motivation for developing a knowledge-centric autonomic framework. These are outlined below, alongside the proposed solutions (relevant to research objectives O1.2 and O1.3).

- A sizeable group of everyday applications operate in realtime (such as server network configuration, where load has to be balanced while maintaining the quality of the provided service, or media display adaptation, where the appearance of text and images needs to be tuned in line with the needs of the audience without any perceptible lag). Under these circumstances, in order to ensure a realistic response time, the system is required to promptly answer questions (to determine the impact of adding a server to the middle layer on the overall load distribution – an example provided in [103] – or the effect that an increase in display brightness will have on the audience's level of attention). One semantic service that is suitable for this task is querying, performed efficiently by all standard reasoner implementations (FaCT++, Pellet, etc.), which invites the use of an ontology to represent the knowledge base of such systems.

- With the increasing popularity of online decision support systems, gaining significant traction in the field of career management support, it becomes necessary for automated online tools to deal with large repositories of incomplete and continuously changing knowledge. In cases where that knowledge is both consulted and curated by the wider community (as opposed to a small group

of domain experts), it is crucial to maintain its consistency (at stake, since community editors are unlikely to be well versed in description logic) as well as display it in a visually intuitive way (for instance, as a graph, where nodes represent professions, rather than a dull database-style table/list). An ontology is based on RDF triples, easily translated from a formal description to a visual representation (concepts correspond to nodes and properties to edges). In addition, the analysis and plan stages of the MAPE control loop can work alongside the reasoner to answer user queries in line with their profiles/preferences and/or in tune with their current circumstances (monitored, for instance, by system sensors).

4.2 Architecture

Fig. 4.1: KAS architecture - component view

The architectural components presented in Fig. 4.1 can be divided in three categories.

- **Passive:** Onto++, Onto++ instance and Policy. Onto++ is a template describing the fixed part of all Onto++ instances that are built from it. Onto++ instances are distinct, real ontologies, modelling specific problem domains, yet laid out in the same structure, namely that of the Onto++ template. The policy is a document containing experiential knowledge authored by the domain expert to guide system behaviour towards a desirable state.

- **Active:** Monitor, Analyse, Plan, Execute, KT and Reasoner. The first four components represent the classic stages of the autonomic control loop [90, 101]. KT (Knowledge Translator) is an ontology learning algorithm that dynamically generates Onto++ instances. The Onto++ template is programmed into KT, but the algorithm will only instantiate those parts that apply to a given

problem domain. The reasoner is a semantic inference engine that maintains the Onto++ instance logically consistent and automatically deduces new, implicit facts from the explicit knowledge generated by KT. The algorithms developed by the author of this research to implement the active components of KAS are presented in 4.3

- **Miscellaneous:** Sensors, Effectors and Managed resource. Strictly speaking, sensors and effectors are active components that read information from the managed resource and, respectively, effect changes onto it. However, relative to KAS, it is only the data produced by the sensors and the actions performed by the effectors that are of interest, not the inner logic of those two components. Similarly, the managed resource may contain active processes that will be captured, to an extent, in the ontology. Yet, KAS does not provide the logic that underpins those processes, although it may indirectly influence it via the changes operated by the effectors. In a nutshell, the complexity of sensors, effectors and the managed resource is abstracted away by the APIs that KAS uses to communicate with those components.

Design justification. The proposed architecture is motivated by *compatibility* with the traditional structure of the autonomic manager [90] and by *portability* to various application domains. Compatibility is ensured by providing implementations for all MAPE modules as well as for the tool generating the knowledge they operate on. Cross-domain applicability is endorsed by making all supporting tools configurable: the methodology (4.4) that integrates all tools controls the way knowledge is generated by KT and used by the autonomic manager in order to meet the requirements of a given application.

4.2.1 Onto++ Template

The fixed part of every Onto++ instance consists in a set of concept and property hierarchy roots shown in the right hand side diagram of Fig. 4.1. The subconcepts and subproperties populating these hierarchies will be partially asserted by KT and partially subsumed (inferred) by the reasoner, to accurately represent specific entities from the modelled domains. Inferred knowledge represents the dynamic part of Onto++ instances and is not contained in the template. The hierarchy roots are presented below.

- `Entity` is the root of the ontology hierarchy modelling the problem domain. `Entity` subconcepts are application specific and are connected to each other via inheritance (e.g., in an Onto++ instance that models a chemical reaction, `OrganicCompund` and `InorganicCompund` directly subsume `Entity`, `Water` inherits from `InorganicCompund`, `Carbohydrate` inherits from `OrganicCompound`, etc.) or via bespoke properties (e.g., `hasMolecule` connects `Carbohydrate` and `Water`). Inheritance relationships are implicit and will be deduced by the reasoner. Bespoke properties are rooted in `hasProperty` and are explicitly asserted by KT. If `Entity` subconcepts have connections to numbers (`hasNumberOfAtoms`), strings (`hasChemicalSymbol`), boolean values (`isStable`), etc., those properties will be asserted under `hasValue`.

- `State` is the base concept for the ontology segment modelling possible system states. If the targeted domain is a chemical reaction, then the state of the system may be defined in terms of the resulting substance's composition (`State1` would correspond to a molecule of one `Carbohydrate` subconcept, `State2` to two molecules of another, etc.). The connections between `State` and `Entity` subconcepts are realised by the `hasEntity` property. Each state has an associated

preference score, called utility, that indicates its desirability from the autonomic manager's perspective. Utilities (block `values` in the architecture) are numbers, not concepts, therefore do not form a hierarchy. States are connected to utilities via the `hasUtility` property.

- `Link` is an abstract concept, in the sense that it does not model a physical entity from the target domain. Instead, it serves as an auxiliary building block to represent multifaceted properties with no natural support in RDF, a process called reification (a detailed example is provided in 4.3.2). The reificated property inherits from `hasLink`. `hasArgOne` and `hasArgTwo` connect the `Link` concept to the domain and filler of the reificated property. The extra facet of the reificated property (a number or a string from the `values` block) is associated to the `Link` concept via property `hasWeight`.

Design justification

The Onto++ structure is designed to meet the autonomic manager's requirements with respect to the representation of the knowledge repository. Specifically, the problem domain should be modelled as a graph [88, 90], where nodes represent physical entities and edges model their connections. These are the roles of the `Entity` and `hasProperty` hierarchies, respectively. The `State` hierarchy (when it is computationally feasible to store it) supports the analysis (4.3.4) and planning (4.3.5) stages of the MAPE loop, whereas the `Link` hierarchy makes it possible to model more complex relationships between entities. Note that these last two hierarchies are included in the Onto++ instances only if they are needed.

Another issue that requires justification is the choice of an ontology for knowledge storage. A case has been made in the introduction (1.1) for the inherent compatibility between semantic technologies (particularly ontologies and reasoners) and autonomic computing, yet a more convincing explanation is required relative to the dismissal of other knowledge management alternatives such as relational databases or object oriented models. The most compelling reason in favour of ontologies is best formulated by Soylu et al: database schemas and object-oriented models "are not meant to capture a domain per se and are not truly natural for end users" [164]. This comes down to these platforms' incapacity to learn, that is, infer implicit knowledge from explicit facts, as a reasoner does with ontology data. Humans implicitly infer knowledge when they experience new environments, therefore semantic models are one step closer to the way organic brains build models of the world. To further strengthen this argument, a comparison between the performance of an autonomic manager storing knowledge in a relational database against that of the same autonomic manager featuring a semantic layer is presented in 5.4.1.

The reason for selecting reification as a means to represent multifaceted properties over other alternatives[1] is twofold. Firstly, a significant portion of the covered literature employs reification in practical applications [169, 35, 66, 60, 134] and secondly, it is recommended as an ontology design pattern [65, 136].

The final argument relates to the absence of plans and policies from Onto++, since other approaches [147, 155, 87, 11] provide semantic representations for those two components as well. Within the KAS framework, policies and plans are not reasoned on. They are generated by the domain expert and, dynamically, by the `plan` algorithm (4.3.5), respectively, but are not subject to semantic inference.

---

[1]See `https://www.w3.org/wiki/PropertyReificationVocabulary#Alternatives` for a W3C endorsed list.

Therefore, including them in the ontology would bring no computational gain. Acording to the covered literature, a consensus is yet to be reached with respect to the cost-benefits ratio of reasoning over plans and policies, thus this approach will be further investigated in future work.

Running example

The very simple smart environment considered for illustration features two cooling devices (a fridge and a freezer) and two room heating appliances (a radiator and an air conditioning unit). These are represented by concepts of the `Entity` hierarchy, in the Onto++ instance shown in Fig. 4.2. Cooling devices have a capacity expressed in litres, whereas heating appliances feature a temperature output in degrees Celsius - these properties are modelled by `hasCapacity` and `hasOutput`, respectively. Both types of machines use power in a specific amount expressed in Watts - this is represented by the `hasWattage` property. All properties mentioned thus far have numeric fillers, therefore their graphical representations in Fig. 4.2 are connected to the `values` box.
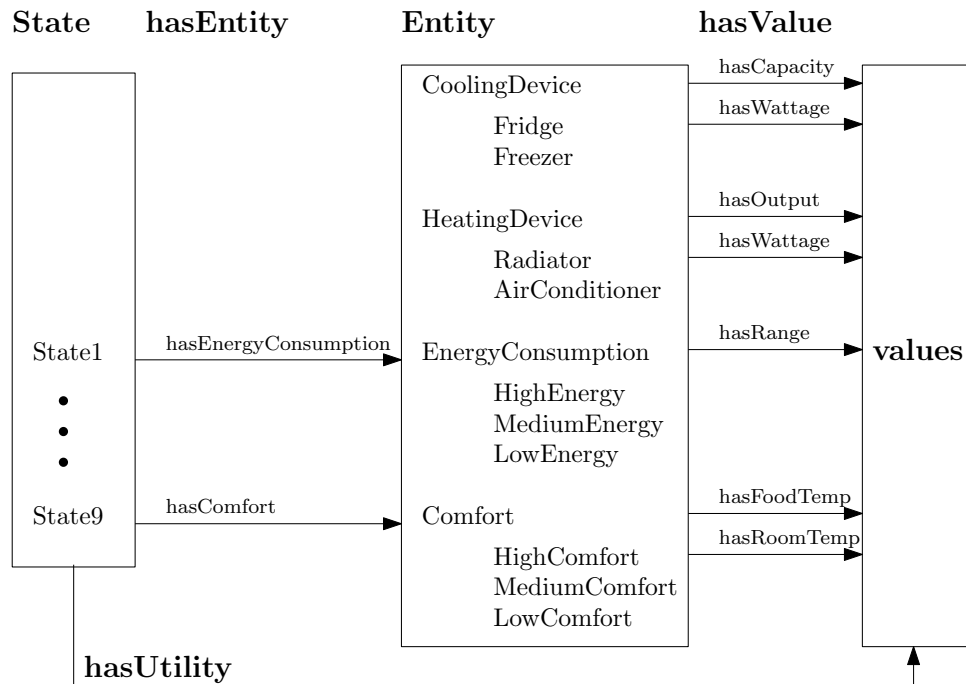
Fig. 4.2: The Onto++ instance modelling the smart environment - concept hierarchies are presented as indented lists

To get a better picture of how the above concepts and properties are connected, the complete definitions of ontology concepts `Fridge` and `AirConditioner` are given in Fig. 4.3 (`Freezer` and `Radiator` can be described in a similar fashion).

The two ontology classes, along with their sub-concepts, towards the bottom of the `Entity` hierarchy, namely `EnergyConsumption` and `Comfort`, support the definition of all possible states that the smart environment can be found in. In this case, there are nine possible states, one for every combination of energy consumption and comfort levels. For instance, `State7` (Fig. 4.4) is reached when the overall energy consumption is low and the level of comfort (room temperature and food preservation) is high. Since maintaining a comfortable ambient temperature and ensuring that kitchen appliances are

```
Class: Fridge EquivalentTo:
  CoolingDevice
  and (hasCapacity value "150"^^integer)
  and (hasWattage value "200"^^integer)
```

```
Class: AirConditioner EquivalentTo:
  HeatingDevice
  and (hasOutput value "20"^^integer)
  and (hasWattage value "2000"^^integer)
```

Fig. 4.3: `Fridge` and `AirConditioner` concept definitions

cool enough to keep food fresh, at a low energy cost, is an ideal situation, concept `State7` represents the goal state and is thus awarded the maximum utility.

```
Class: State EquivalentTo:
  hasEnergyConsumption some EnergyConsumption
  and (hasComfort some Comfort)
  and (hasUtility some xsd:integer)
```

```
Class: State7 EquivalentTo:
  State
  and (hasEnergyConsumption some LowEnergyConsumption)
  and (hasComfort some HighComfort)
  and (hasUtility value "9"^^integer)
```

Fig. 4.4: `State` and `State7` concept definitions

The values used to define the `EnergyConsumption` and `Comfort` subclasses (in other words, the range a temperature value would have to be in to be considered high) are either prescribed manually or discretised automatically (e.g., when given the upper and lower limits of the energy consumption value range – for instance, 1000W and, respectively, 4000W – and the number of desired classes – say, 3 – the manager will automatically determine that a high energy consumption is over 4000W, a medium one is between 1000W and 4000W and low one is under 1000W). To illustrate, the definitions of `HighEnergy` and `MediumComfort` are given in Fig. 4.5.

Concepts `EnergyConsumption` and `Comfort` (specifically, its two facets, room and food temperature) correspond to the environment signals that the autonomic manager is capable of monitoring. Provided that a new sensor is installed to measure, for instance, humidity, a new ontology class will be asserted with the appropriate sub-hierarchy (for a discretisation factor of 2, sub-concepts `LowHumidity` and `HighHumidity` would be considered).

4.2.2 Policy

The KAS policy document is created by the domain expert and has three sections:

```
Class: HighEnergy EquivalentTo:
  EnergyConsumption
  and (hasRange some xsd:integer[>4000])
```

```
Class: MediumComfort EquivalentTo:
  Comfort
  and (hasFoodTemp some xsd:integer[>3 , <=5])
  and (hasRoomTemp some xsd:integer[>15 , <=20])
```

Fig. 4.5: `HighEnergy` and `MediumComfort` concept definitions

- **utilities.** These are numerical values associated to system states. In the most common scenario, the system state is given by the values of the monitored inputs. For example, one of the states that a system monitoring temperature and humidity may be in is described by:

  ```
  temperature <= 10 AND humidity <= 100   5
  ```

  In this example, a utility of 5 is associated to the state where the environment temperature is lower than 10 units and the humidity is under 100 units. All relevant system states must be provided with a utility in this section of the policy document. Input discretisation (a task performed during ontology learning - see 4.3.2) is directly relevant to utility definition as it significantly reduces the size of the state space. Unless otherwise specified, the autonomic system **goal** is defined as achieving the state with the highest utility.

- **actions.** This section of the policy document is related to the managed resource parameters modified by the autonomic system's effectors. Relative to the previous example, let us assume that the controlled parameter is the environment's pressure, leading to three possible actions: increasing, maintaining or decreasing it. At each step of the MAPE loop's planning stage, an action is selected from the available set and inserted in the plan under development.

- **heuristics.** These represent high level knowledge prescribed by the domain expert to deal with safety-critical situations (like in the example to follow) or help expedite the analysis and planning stages of the MAPE loop (see 5.2.3). An example of a heuristic policy expressed in the ECA formalism is given below:

When the temperature is above 150	*event*
AND	
the pressure valve is faulty	*condition*
THEN	
switch off power.	*action*

All policy sections are optional apart from the goal definition (given by the maximum utility state as in the example above or by a *goal* policy as in 6.2).

Running example

In the case of the smart environment application, the policy document would contain:

- utility values for all nine states (a low level of comfort obtained at a high energy cost is intuitively the minimum utility state, just as a high level of comfort at a low energy cost – Fig. 4.4 – is the system goal and, consequently, the highest utility state; however, it takes a human system administrator to assign utilities for less clear-cut states, as, in some scenarios, high comfort at a medium energy cost may be preferable to medium comfort at a low energy cost, whereas, in others, the situation may be reversed)

- actions that the autonomic manager can operate on the environment (for example, switching appliances on or off and commuting to a low power state or back to regular operation parameters)

- heuristics such as the one below.

When the energy consumption is medium	*event*
AND	
the comfort level is high	*condition*
THEN	
switch all electrical appliances over 200L to low power mode.	*action*

4.2.3 Managed Resource, Sensors and Effectors

The managed resource is a hardware and/or software system (such as the chemical reaction in the example above, assuming that the reactants are added by dedicated, programmable hardware) that the autonomic manager controls. The legacy system is capable of functioning independently from the manager, only not as efficiently. The clear separation between the managed resource and the manager is an important criterion when investigating whether an applications is truly autonomic (for this reason, the managed resources of the two systems that implement KAS will be clearly outlined). Similar to the negative feedback loop in classic control theory, the autonomic manager reads some parameters (e.g., the molecular composition of the compound produced by a certain chemical reaction) from the managed resource, via sensors, and controls other parameters (e.g., the amount of light that the reaction is exposed to), via effectors. The purpose of the control process is to drive the managed resource to the goal state (a certain chemical composition of the resulting substance that the domain expert assigns the maximum utility to).

4.3 Tools

4.3.1 Reasoner

KAS makes use of FaCT++ [175], a reasoner implemented in C++ and supported by the OWL API (`http://owlapi.sourceforge.net/`). The reasoner features exploited by KAS are:

- **subsumption.** This is the logical operation of automatically determining the superclass of a newly asserted concept. To illustrate, consider a scenario where the time of day is a

monitored input, updated every hour. At 9 am, a new ontology concept will be created, namely `crtTime hasHour 9`. Based on that definition, the reasoner will *infer* that `crtTime` is a subconcept of `earlyMorning hasHour [>=9 <11]`. Thus, the reasoner automatically concluded that 9 am is an hour in the early morning (and not in the late afternoon), which represents a new piece of domain knowledge. Subsumption supports KAS's ability to learn new facts about its environment, even those that human experts **did not** explicitly programme into the problem domain description.

- **classification.** The reasoner automatically verifies the logical correctness of all ontology axioms. In case the ontology contents is extracted from a legacy description (e.g., a file listing system entities that were manually compiled by a domain expert), the classification operation will flag up logical contradictions such as `timeStamp (hasTime earlyMorning) and (hasHour 14)` by declaring concept `timeStamp` unsatisfiable.

- **sensor data verification.** This is a consequence of classification. Given that the ontology concepts modelling the system's state are defined in a logically robust way (see 5.2.2 for details), the reasoner will detect an illegal input value such as hour -1.

- **cycle detection.** This feature stems from subsumption. A cycle is a circular "is_a" relationship connecting two or several ontology concepts (an example of a cycle with three concepts is "A is_a B, B is_a C, C is_a A"). This situation is very likely, especially when learning the ontology from a large, manually curated legacy description. A cycle is not a logical error per se, however, the reasoner will deduce that the concepts involved are equivalent (in the previous example, A ≡ B ≡ C). A software tool can be set up to retrieve all equivalence relationships in the ontology and investigate whether they represent a cycle or not.

- **querying.** This is useful for searching the ontology for a specific concept and its direct neighbours. For that purpose, a DL query is built, namely a logical definition that all concepts being searched for have to meet (query `State and hasTime EarlyMorning` will match all subclasses of `State` connected via the `hasTime` property to concept `EarlyMorning` or one of its subclasses). The DL query can be viewed as an anonymous ontology concept subsumed by all the classes that meet the search criteria.

The use of the reasoner in terms of supporting autonomic decision making will be illustrated on the running example at the end of section 4.3.4, as it is tightly coupled with the analyse and plan stages of the MAPE-K loop.

4.3.2 Knowledge Translator

The Knowledge Translator (KT) is an ontology learning algorithm (Table 4.1) that automatically generates an ontology from an initial repository of domain knowledge.

Inputs and Outputs

Besides the initial knowledge repository, KT takes two other inputs, a boolean flag to distinguish between discrete ant continuous data and a collection of discretisation thresholds to deal with the latter data type.

Table 4.1: The Knowledge Translator (KT) algorithm

KT (inputFile, isDiscrete, thresholds) **returns** o

1	Entity, State, Link = {}
2	hasEntity, hasProperty, hasLink, hasArgOne, hasArgTwo, hasWeight, hasUtility, hasValue = {}
3	
4	**if** isDiscrete = **false then**
5	**for each** t **in** thresholds **do**
6	e = assertConcept(Entity);
7	**if** hasWeight(t) = **true then**
8	reificate(e, t, getWeight(t));
9	**else**
10	assertProperty(hasValue, e, t);
11	**end if**
12	**end for**
13	**else**
14	**for each** [domain, filler, prop] **in** inputFile **do**
15	e1 = assertConcept(Entity, domain);
16	e2 = assertConcept(Entity, filler);
17	**if** hasWeight(prop) = **true then**
18	reificate(e1, e2, getWeight(prop));
19	**else**
20	assertProperty(hasProperty, domain, filler);
21	**end if**
22	**end for**
23	**end if**
24	
25	**for each** utility **in** inputFile **do**
26	s = assertConcept(State);
27	assertProperty(hasUtility, s, utility);
28	**for each** e **in** getDirectSubclasses(Entity) **do**
29	assertProperty(hasEntity, s, e);
30	**end for**
31	**end for**
32	
33	o = [Entity, State, Link,
34	hasEntity, hasProperty, hasLink, hasArgOne, hasArgTwo, hasWeight, hasUtility, hasValue];

The return value of KT is the generated ontology. The algorithm's inputs and output are detailed in the following.

Input `inputFile`. This represents the problem knowledge repository. It contains the names of domain entities and those of the propeties that connect them. A typical row in the `inputFile` is `entityName1 propertyName entityName2 propertyWeight`, describing two entities in the problem knowledge space, namely `entityName1` and `entityName2` linked by property `propertyName` with a weight given by `propertyWeight`. Entity `entityName1` is refered to as the property domain, entity `entityName2` is the property filler and weight `propertyWeight` is most commonly the probability or the strength of the property. For instance, `Student hasDegree ComputerScience 95` could be interpreted in plain English as "A student will graduate from a Computer Science course with

Table 4.2: Link hierarchy and associated properties construction

reificate(e1, e2, weight)

1	l = assertConcept(Link);
2	assertProperty(hasLink, e1, l);
3	assertProperty(hasWeight, l, weight);
4	assertProperty(hasArgOne, l, e1);
5	assertProperty(hasArgTwo, l, e2);

a probablity of 95%". The `inputFile` may also contain utilities, namely numbers associated to each possible state of the system. These are usually extracted from the policy document.

Input `isDiscrete`**.** The data in the `inputFile` can refer to discrete entities in the knowledge space, such as `Student` and `ComputerScience` in the previous example, or continuous inputs, for example the time of day or the level of a laptop's battery. In the former case, corresponding to `isDiscrete` being true, KT will build the ontology by extracting `inputFile` data and directly translating it to concepts and properties. In the latter case, KT will automatically discretise the continuous inputs (ususally provided by the autonomic manager's monitor) into categories of values and assert each one as a concept in the ontology.

Input `thresholds`**.** An array of numbers, representing cut-off limits for defining dicrete categories of continuous inputs' values, that is ignored when `isDiscrete` is true, but must be provided in the opposite case.

Output `o`**.** This is an instance of the Onto++ template. It contains all components illustrated in Fig. 4.1: `Entity`, `State` and `Link` are initially empty concept hierarchies (containing just the base eponymous classes), whereas `hasEntity`, `hasLink`, `hasArgOne`, `hasArgTwo`, `hasWeight`, `hasUtility` and `hasValue` are property hierarchies, initially holding the base property only. The concepts linked by these properyies are of the types depicted in 4.1 (where the arrows representing the properties point from the ontology class representing the property domain to the one that stands for the property filler). Hierarchies `hasEntity`, `hasProperty` and `hasLink` contain object properties, that is, both their domains and their fillers are ontology concepts. Hierarchies `hasWeight`, `hasUtility` and `hasValue` comprise data properties, where the domain is a concept and the filler is a numerical value. Properties subsuming `hasArgOne` and `hasArgTwo` can be either object properties, when the input file contains discrete data, or data properties, in the case of continuous data that has been discretised. Since the former case is most common in automatic applications, `hasArgOne` and `hasArgTwo` are represented as object properties in Fig. 4.1.

Running Example

To aid the reader with understanding the algorithm logic presented in the following section, this is what the KT algorithm inputs and output would be, in the case of the smart environment application.

Input `inputFile`**.** The first lines, used to assert the `CoolingDevice` sub-hierarchy in Fig. 4.2, are given below (no weights are assigned, as this example does not require multi-faceted properties). The rest of the input file would follow a similar template.

CoolingDevice	hasCapacity	[0 200]
CoolingDevice	hasWattage	[50 5000]
Fridge	hasCapacity	150
Fridge	hasWattage	200
Freezer	hasCapacity	100
Freezer	hasWattage	250

Inputs `isDiscrete` and `thresholds`. The boolean is `false`, as the monitored signals (wattage and temperature) are continuous. Therefore, the thresholds passed in to discretise overall energy consumption, room temperature and food storage temperature, are, respectively: [1000 4000], [15 20] and [3 5].

Output `o`. This is the Onto++ instance presented in 4.2. Note that there is no `Link` hierarchy, as that is only relevant in the process of reification (illustrated in chapter 6).

Algorithm Logic

The KT algorithm has three main parts, each responsible with the creation of one concept hierarchy in the ontology along with the associated properties. Those parts are described in the following, starting with a short note about the pseudo-code in Table 4.1.

Notations. Method `assertConcept` (lines 6, 15, 26) creates and returns a new class in the hierarchy given as parameter. That is to say that `e = assertConcept(Entity)` creates a new subclass (referred to, later in the algorithm, as `e`) of `Entity`. Method `assertProperty` (lines 10, 20, 27, 29) creates a new property, in the hierarchy given as the first parameter, connecting the domain given as second parameter to the filler given as third parameter. That is to say that `assertProperty(hasValue, e, t)` creates a new subproperty of `hasValue` connecting previously asserted class `e` to numerical value `t`. Method `assertProperty` has no return value. Although omitted from the psuedo-code for simplicity, all `assertConcept` and `assertProperty` methods may take an extra parameter holding the name of the new class/property. If that is missing, the name will be generated automatically.

`Entity`**,** `hasProperty` **and** `hasValue` **hierarchies creation** (lines 4 - 23 in Table 4.1)**.** If the system is monitoring continuous data, the values in the `thresholds` array will be used to discretise it. One `Entity` subclass will be asserted for each of the resulting discrete categories. For example, if the `thresholds` array holds values 10, 20 and 30, associated to monitored input **M1**, KT will assert concepts `M1_1` (line 6) connected via `hasValue` to interval [10 20] (line 10) and `M1_2` linked to [20 30] by means of the same property. It is possible to specify the names of the new `Entity` concepts to replace the default generated ones (for simplicity, the extra name parameter was excluded from the pseudo-code). It is also allowed to construct more complicated ranges of values for each discrete category corresponding to a monitored signal (an example using logical operators is used for the SAR system in chapter 5). If the data read in from the `inputFile` is discrete (presented as a sequence of rows, each containing the domain, filler and name for a given property - line 14), KT will directly translate it in the appropriate ontology elements (lines 15, 16, 20).

`Link`**,** `hasArgOne`**,** `hasArgTwo`**,** `hasWeight` **and** `hasLink` **hierarchies creation** (Table 4.2)**.** It is common for properties to have several facets. For instance, the polar covalent bond between a water

moecule's hidrogen atoms and its oxygen one can be described by property `hasBond` with domain `Hydrogen` and filler `Oxygen`. To make for an accurate model, the property would need two extra facets: one stating that the bond is `covalent` and another describing the bond as `polar`. Unfortunately, this goes beyond the modelling possibilities of RDF, making it necessary to introduce the extra facets via reification [166]. This ontology design pattern implies inserting an extra concept on the connection path between the domain and the filler, as illustrated in Fig. 4.6. KT implements a similar process (lines 8 and 20) by calling the `reificate` method described in Table 4.2. There, the extra concept introduced on the connection path between e1 and e2 is 1, a subclass of `Link`, the multifaceted property is asserted in the `hasLink` property hierarchy and the extra facet is `hasWeight`.
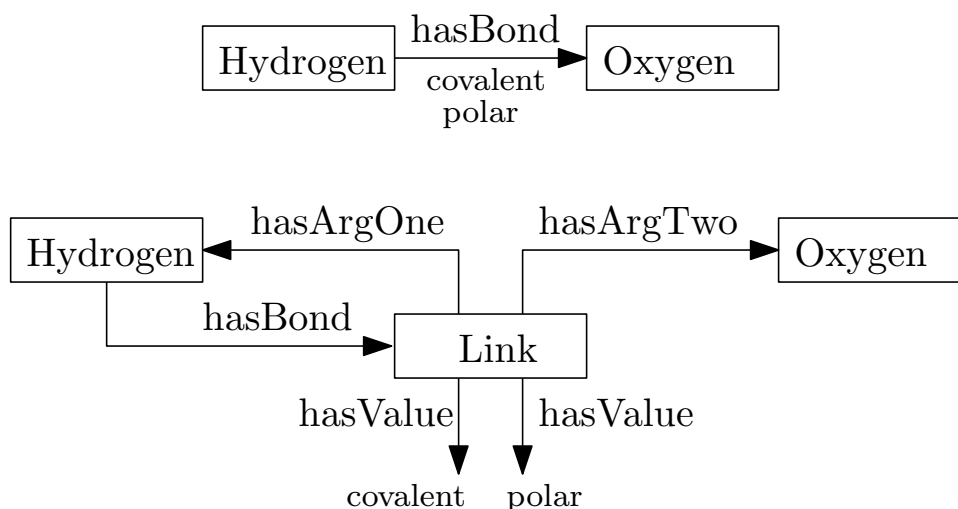
Fig. 4.6: Reification example

`State,` `hasUtility` **and** `hasEntity` **hierarchies creation** (lines 25 - 31 in Table 4.1)**.** A new concept is created in the `State` hierarchy for every possible combination of concepts in the `Entity` hierarchy. To exemplify, consider a simple `Entity` hierarchy with two subclasses: `Absenteeism` and `Progression`. After discretisation, each subclass has been provided with two categories: `LowAbsenteeism, HighAbsenteism` and `LowProgression, HighProgression`, respectively. It follows that the system described by this `Entity` hierarchy may be in one of four states, each associated with a utility (a number measuring the "desirability" of the state), as shown in Fig. 4.7. Assuming that the modelled system is a university, low student absenteeism and high student progression is the preferred state, hence the associated utility will be the highest of the four.

Should the number of `Entity` concept combinations be too high, a simplified `State` hierarchy (where irrelevant entity combinations have been eliminated) may be explicitly added by the ontology user. It is also possible to omit the `State` hierarchy altogether, however that will impact the functionality of the autonomic manager (4.3.4). The utility associated to each `State` class is extracted from the `inputFile`.

Advantages

The key benefits of KT are discussed below.

Automated ontology learning. KT extracts knowledge from existing descriptions of the managed resource (the system to be controlled by an autonomic manager) and stores it into an ontology, with

Fig. 4.7: `State` hierarchy example - leaf level classes and properties only

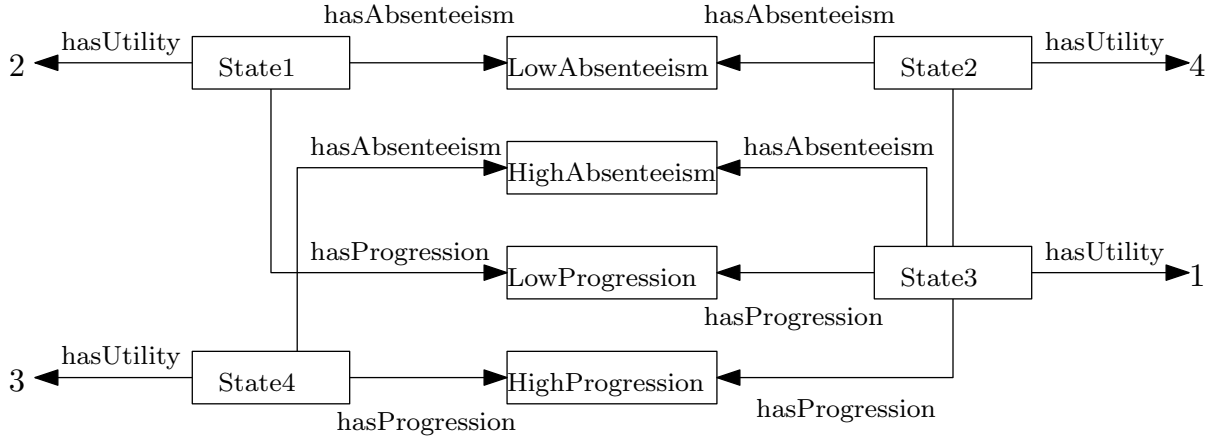limited or no human intervention. This approach also enables the semantic processing (4.3.1) of that knowledge, potentially generating more insight than what could be obtained from traditional knowledge models (such as databases).

Discretisation. Signals monitored by autonomic managers are usually continuous. Analysing continuous data models (e.g., time series, continuous probability distributions etc.) can be computationally expensive [88, 103]. To help speed up the autonomic analysis phase (4.3.4), KT automatically discretises continuous inputs in several categories of values, given some provided thresholds.

Reification. To overcome a limitation of RDF representation, KT applies an ontology design principle called reification. This adjusts the semantic hierarchies to allow the modelling of multi-faceted properties, ultimately allowing a more powerful autonomic analysis process. This comes at a cost, namely that of increasing the ontology size.

System state modelling. Autonomic analysis and planning rely heavily on analysing system state [88] (in order to compare the current one against the goal state, extract patterns of previous system behaviour or even detect sensor anomalies). To facilitate analysis and planning, KT provides support for automatically generating and storing the complete set of system states.

Polynomial complexity. The complexity of KT is:

$$O(E * L + \prod_{i=1}^{m} E_{M_i}) \tag{4.1}$$

where E is the number of concepts in the `Entity` hierarchy, L represents the number of `Link` classes and E_{M_i} stands for the number of `Entity` subclasses associated to monitored input M_i. In terms of KT inputs, E is given by either the size of the `thresholds` array (if `isDiscrete` is false) or by the number of rows in `inputFile` (if `isDiscrete` is true). L is the number of weights associated to either each treshold or each property on an `inputFile` row. The final term of the addition in (4.1) represents the number of `State` concepts. In the example in Fig. 4.7, there are two monitored inputs: absenteeism (M_1) and progression (M_2), each discretised in two classes. Thus, E_{M_1} and E_{M_2} are both 2, leading to a `State` hierarchy with 4 concepts.

4.3.3 Monitor and Execute

The **monitor** module is an interface between the sensors reading information from the managed resource and its environment, on the one hand, and KAS's autonomic manager, on the other hand. It samples the monitored signals at prescribed intervals of time and passes the values either to the ontology (in order to maintain the `State` hierarchy) or directly to the analysis component (if the set of system states is too complex to represent within the ontology). When deploying an instance of KAS in a specific application domain, a bespoke monitor implementation must be provided to read data from physical devices such as cameras or "software sensors" such as graphical user interfaces.

The **execute** component provides a shell for implementing planned actions (such as `increase font` or `decrease brightness`) within the managed resource (for example, an electronic document rendered in a text editor). This module requires a problem-specific implementation (most likely reliant on the appropriate APIs to control system effectors).

4.3.4 Analyse

The analysis algorithm (Table 4.3) identifies the system's current state and uses the high level heuristics in the policy document to refine the set of available actions for the planning phase.

Table 4.3: The analyse algorithm

analyse (o, policy) **returns** (u, actions)

1   s = findCrtState();
2   u = getFiller(s, hasUtility);
3   actions = applyHeuristics(policy, s);

Inputs and Outputs

Input `o`. The system ontology may contain a `State` hierarchy with concepts representing each relevant state of the system. During analysis, the `State` subclass modelling the current system state will be identified. In case the `State` hierarchy is absent from the ontology, analysis will be based on raw sensor input (see 6.2.5).

Input `policy`. The heurisics defined by the domain expert as part of the policy document are used during analysis to discard some of the actions to be considered during planning.

Output `u`. The utility associated to the current system state is determined and returned by the analysis algorithm.

Output `actions`. The final set of actions to be considered during planning (after eliminating the ones that do not comply with the policy heuristics) is also determined during analysis.

Algorithm Logic

Current state identification (line 1). Every time a new set of input values is read by the monitor, method `findCrtState` will identify the `State` subclass that matches those values. With

reference to the example in Fig. 4.7, if the monitored values are subsumed under `LowAbsenteeism` and `HighProgression`, the current state of the system will be identified as `State2`.

Current state utility retrieval (line 2) **.** Method `getFiller` takes two parameters, namely an ontology concept and one of its properties. Here, the former is the class representing the current system state, whereas the latter is the `hasUtility` property (Fig. 4.1). The returned value is the filler of that property, in this case, the utility associated to the current state. In in Fig. 4.7, the utility of `State2` is 4.

Action set construction (line 3) **.** The action segments of relevant policy heuristics will be consulted to form a set of plan actions (that the planning component will use to formulate plans). Specifically, method `applyHeuristics` will find all `policy` heuristics where the event and condition segments match the current system state, `s`. The action segments of these heuristics will be executed to eliminate non-conforming plan actions from the second section of the policy document (4.2.2), resulting in a simplified `actions` set that the method returns. Relative to the example heuristic provided in 4.2.2, all policy actions apart from `switch off power` will be excluded from the set of candidate plan actions.

Running Example

With reference to the smart environment application, the inputs, logic and outputs (presented in that order for continuity) of the analysis algorithm are explained in the following.

Input `o`. This is the Onto++ instance, illustrated in Fig. 4.2, that features a `State` hierarchy.

Input `policy`. The utilities, actions and heuristic given in the running example section in 4.2.2 are passed into the analyse algorithm.

Algorithm logic. Let us assume the monitor detects average temperatures of 20 degrees Celsius in the rooms of the smart home and of 2 degrees Celsius in the cooling devices where food is kept. This will trigger the assertion of concept `CurrentComfort`, with the numerical fillers above, which the reasoner will subsume under `HighComfort`. The measured overall energy consumption, at the current sampling moment, is 1000W. Consequently, concept `CurrentEnergyConsumption` is asserted with the appropriate filler and subsumed under `MediumEnergyConsumption`. Classes `HighComfort` and `MediumEnergyConsumption` are the fillers of `State4`, which is thus detected as the current system state (see 4.2.1 – the running example paragraph – for the full concept definitions). This situation matches the event and the condition of the heuristic in the policy document, therefore the analysis algorithm will filter out all other actions save powering down all devices with a capacity greater than 200L.

Output `u`. In this case, th utility associated with state 4 will be retrieved from the policy document and returned.

Output `actions`. The set will solely contain the action of the only heuristic in the policy document.

Advantages

Simplicity. According to the general IBM specifications [90], the second MAPE stage is responsible for matching sensor data against known models of the system state and using the results of that analysis to inform the planning process. In the KAS architecture, the state model is stored in the ontology and much of the complexity of comparing sensor data against that model is delegated to the reasoner. Specifically, via subsumption (4.3.1), the reasoner will automatically infer the correct categories that the latest inputs from the managed resource belong to (e.g., 9 am is `EarlyMorning`).

The analysis component is thus left with the simpler task of matching those categories against the filers of `State` properties (find the `State` subclass linked to `earlyMorning`). Consequently, the application developer need not be an expert in Markov chains [34], time series [182] or other complex mathematical modelling approaches in order to implement the anaysis component. This can be performed with an off-the-shelf reasoner and the algorithm in Table 4.3.

Polynomial complexity The complexity of the analyse algorithm is:

$$O(m + S + h * a_h) \tag{4.2}$$

where m is the number of monitored inputs and S denotes the number of `State` classes in the ontology. The sum of these two terms represents the computational complexity of line 1 in Table 4.3. Line 3 is executed in $h * a_h$ steps, where h is the number of policy heuristics and a_h stands for the number of candidate plan actions to be excluded by executing each heuristic.

4.3.5 Plan

The KAS planning algorithm is described in Table 4.4 and is responsible with dynamically creating plans (sequences of actions) to execute in response to changes in monitored data. An efficient plan will improve the utility of the system's state, ideally to the point of reaching the maximum utility.

Table 4.4: The plan algorithm

plan (p, planBank, u, actions) **returns** p

1	q = getPlan(planBank, u);
2	**if** q != {}
3	p = p ∪ q;
4	**else**
5	p = p ∪ selectAction(actions);
6	**end if**

Inputs and Outputs

Input and output p. A plan $p = \{a_i, i = 1..N\}$ is a sequence of N actions a_i selected from the set available in the policy document. The current plan is developed by dynamically adding new actions or entire known good plans from the plan bank.

Input `planBank`. This is a collection of efficient plans developed by the plan algorithm throughout the operation of the KAS instance. Formally, the plan bank is a set of triples $\{(p, u_0, u)\}$, each representing a plan p that evolves the system state from utility u_0 to utility u, where $u > u_0$.

Input `u`. This represents the utility of the current system state and is provided by the analyse algorithm.

Input `actions`. Also an output of the analyse algorithm, this is the set of eligible actions remaining after the consideration of policy heuristics.

Algorithm Logic

Output `p`, namely the current plan, is constructed by adding actions from the avilable set or entire known good plans from the plan bank. The algorithm in Table 4.4 describes one step of the process. For the complete creation of a plan, the algorithm will run in a loop as shown in 4.4.

Plan bank consultation (lines 1 - 3)**.** If the bank contains a plan `q` that, during the system's previous operation, proved successful at increasing initial utility `u`, all of `q`'s actions will be added to current plan `p`.

Dynamic plan construction from action set (line 5)**.** Alternatively, if the bank contains no suitable plan for the given initial utility, an action will be selected from set `actions` and added to plan `p`, currently under development.

Running Example

The inputs, logic and output of the plan algorithm, including some discussion around connections with the analysis phase, are presented below, in the context of the smart environment application.

Inputs `p` and `planBank`. Let us assume that, initially, the action in the current plan is the one extracted from the policy heuristic (according to the logic explained in 4.3.4 – running example paragraph). The plan bank starts out as an empty set.

Input `u`. The utility of the current system state is the one associated to state 4 (medium energy consumption and high comfort).

Input `actions`. The full set of actions from the policy document is passed into the plan algorithm.

Algorithm logic. As the plan bank is empty (there is no previously executed sequence of actions that successfully drove the system from state 4 to a state of higher utility), a random element will be selected from set `actions` and added to plan `p`. As explained in the KAS methodology section (4.4), the analyse algorithm takes over at this point and deploys candidate plan `p`. If the ensuing system state matches the system goal (state 7), the current MAPE iteration stops and `p` is included in `planBank`. If that is not the case, but the resulting state is of higher utility than that of state 4, the plan algorithm is run again in an attempt to improve `p` by adding another action. Finally, if the initial utility drops, `p` is discarded and the action selection process re-commences.

Output `p`. The sequence of actions that increases the system state's utility will be returned by the plan algorithm.

Advantages

Learning from experience. Plan composition is the process of integrating a known good plan in the current one (lines 1 - 3 in Table 4.4). The current plan will be afterwards improved, if possible, by adding other available actions, in an attempt to further increase the utility of the final system state (this becomes apparent when the plan algorithm is integrated in the KAS methodology, as shown in 4.4). Recycling plans is another facet (besides the reasoner's subsumption feature) of the process of learning, this time from previous experience, as implemented by KAS.

Lightweight storage. The plan bank is stored outside the ontology (in a Java object model) thus eliminating the overhead implied by reasoning on plans [146]. This is beneficial in terms of reducing the runtime of the MAPE control loop.

Polynomial complexity The computational complexity of the plan algorithm is given below.

$$O(P + max(N_q, N))$$ (4.3)

where P is the number of plans in the bank, N_q represents the number of actions contained by bank plan q (if such a plan exists) and N stands for the length of the `actions` set.

4.4 Methodology

The components in the KAS architecture (Fig. 4.1) along with the tools supporting them (described in 4.3) are to be integrated as shown in the general KAS methodology (Table 4.5). The algorithm is meant to be used by application developers as a guide to implementing KAS instances in various application domains (two of which are illustrated in 5 and 6).

Table 4.5: The KAS methodology

KAS()

1	policy = createPolicy();
2	inputFile = createInputFile();
3	o = KT(inputFile, isDiscrete, thresholds);
4	
5	planBank = {};
6	**while true**
7	o = monitor(sensors);
8	(u_0, actions) = analyse(o, policy);
9	
10	u = u_0; p = {};
11	**while** u < u_G **and** size(p) < L **and** actionsLeft = **true**
12	p = plan(p, planBank, u, actions);
13	execute(p, effectors);
14	o = monitor(sensors);
15	(u, actions) = analyse(o, policy);
16	**end while**
17	
18	**if** u_0 < u
19	planBank = planBank $\cup$ (p, u_0, u);
20	**end if**
21	**end while**

Algorithm Logic

Setup and ontology learning (lines 1 - 3). Before running the actual MAPE cycle (the "infinite" while loop on lines 5 - 22), two preliminary tasks need to be performed, namely defining the policy document and creating/preparing the legacy description of the managed resource. Method `createPolicy` on line 1 implies formulating the utilities, actions and heuristics described in 4.2.2, an operation performed by the domain expert. Method `createInputFile` (line 2) generates or retrieves

the legacy description (authored by the domain expert) of the managed resource and adapts its format to the one accepted by KT. This last task is performed by the application developer (the one implementing a KAS instance to fit a specific problem domain) and should ensure that every row in the `inputFile` describes one entity of the managed resource, alongside its properties, their fillers and weigths (if any) - an example of an `inputFile` row was provided in the inputs description in 4.3.2. Once all KT input parameters have been provided and appropriately formatted, the initial ontology is automatically extracted (line 3) from the `inputFile` according to the ontology learning algorithm described in Table 4.1.

Monitoring and analysis (line 7 - 8). The necessary APIs for capturing physical sensor feeds and turning them into numeric data need to be set in place by the application developer. If the ontology stores the system state, the monitor component (4.3.3) will automatically assert (line 7) new ontology concepts to represent the data read in from each sensor. If the number or relevant system states is too high to store in the ontology, the monitored data will be passed directly to the analysis module. Here, the utility of the system's current state along with a (potentially) simplified set of actions is made available (line 8), in an unsupervised way, for use during the planning stage.

Planning (lines 11 - 16). After each incremental addition to the current plan p (line 12, where the method being called is described in Table 4.4), the actions it contains are executed (line 13) via the effectors (programmed by the application developer). The system's response is monitored and analysed (lines 14 - 15) in order to determine whether the utility u of the new system state is closer to that of the goal state, u_G. This condition alongside a size restriction (namely the number of actions in plan p being capped at L) constritutes the planning termination criterion (line 11).

Maintaining the plan bank (lines 18 - 20). If plan p drives the system to a state with a utility higher than the initial one, then it will be stored in the repository of known good plans, `planBank`, for later reference. Plan p will replace any existing plan in the bank associated to the same initial utilty u_0 but with a lower final utility.

Advantages

On top of the advantages of each KAS tool (KT - 4.3.2, monitor and execute - 4.3.3, analyse - 4.3.4, plan - 4.3.5), some new benefits become visible by analysing the general KAS methodology.

Required expertise. Most steps of the KAS methodology are (semi-)automated. The most costly manual operations are porting the managed resource's legacy description into a KT-compatible format and explicitly programming the monitor/execute APIs. However, the computationally intensive stages of KAS operation (ontology learning, analysis and planning) are fully automated. A complete list of actors (stakeholders) responsible for each KAS phase is provided in Table 4.6. The domain expert is a person intimately familiar with the business rules as well as the full set of requirements the KAS instance needs to meet in a given application domain. By authoring the policy document (especially the heuristics - 4.2.2), the domain expert communicates his/her insight in a non-technical language. The application developer is a technical professional, with the necessary skills to translate data between different formats, implement software APIs for interfacing various KAS components and define algorithm inputs (such as KT's `thresholds`). In some situations the input of both the domain expert and the application developer is needed, for instance, when defining parameters like L, the upper bound for the number of actions in a plan.

Table 4.6: KAS methodology - stage authors

Stage	Line in Table 4.5	Author
Policy definition	1	domain expert
Input file creation	2	domain expert
Input file formatting	2	application developer
Ontology learning	3	automated
sensor APIs	7, 14	application developer
monitoring	7, 14	automated
analysis	8, 15	automated
planning	12	automated
effector APIs	13	application developer
execution	13	automated
plan bank management	19	automated

Semantic-autonomic cooperation. The components of the autonomic manager are supported by semantic technologies. Storing the managed resource description in an ontology allows for a more flexible model, in that limitations such as databases' closed world assumption [129] are not applicable to semantic representations and design patterns such as reification [65] are available to represent weighted properties. The reasoner directly supports ontology learning (concept hierarchy management via subsumption), monitoring (sensor data verification) and analysis (subsumption facilitates the retrieval of current system state).

4.5 Evaluation

The Architecture Tradeoff Analysis Method (ATAM) [100] is the leading IT system architecture evaluation technique in modern software engineering. It provides a well structured procedure to evaluate an architecture's fitness taking into account a set of non-dominant quality attributes (in the sense that improving one attribute will implicitly worsen another). ATAM has been applied to refine the KAS architecture as shown in the following.

4.5.1 Problem Definition

An autonomic architecture is needed to control a range of legacy IT systems from different domains. Two specific legacy applications will provide the initial test cases to help shape the architecture and deploy/evaluate/improve it in realistic scenarios. Those applications are: self-adaptive document rendering (SAR) and career decision support (CDS), introduced in (1.3.1) and (1.3.2), respectively.

The stages of ATAM described next rely on input from project stakeholders present in the meetings. The stakeholders in attendance for KAS evaluation were:

- clients: Dr Hai Wang, Aston University (SAR) and Lord Ralph Lucas, Lucas Publishing and Good Careers Guide (CDS)

- domain expert: Lord Ralph Lucas (CDS)

- project managers: Alina Patelli, Aston University (SAR), Prof Ian Nabney, Aston University (CDS)

- developers: Alina Patelli (SAR and CDS), David Bennett (CDS)

- technical consultant: Dr Hai Wang - on semantic technologies (CDS).

4.5.2 Business Drivers

In the SAR domain, the architecture would have to enable *unsupervised*, *realtime* operation. Specifically, the way the document is rendered should adapt to variations in the monitored inputs without human calibration nor significant lags. In the CDS domain, the architecture should allow heterogeneous information *integration*, intuitive *navigation* of that information and *personalisation* of reports and search results.

4.5.3 Architectural Plan

The proposed architecture is an ontology supported autonomic manager. It addresses the previously identified business drivers in that autonomic managers are specifically designed for *unsupervised*, *realtime* operation, ontologies store heterogeneous information uniformly (thus supporting *integration*) as semantic graphs (intuitively *navigable* structures) and reasoners support semantic querying, a powerful instrument for providing insightful, *personalised* results.

4.5.4 Architectural Approaches

To verify the robustness of the plan, the following alternatives were investigated with respect to the architecture (**A**) of the autonomic manager as well as the type (**T**) and structure (**S**) of the knowledge layer.

A1 hierarchical vs flat autonomic system

A2 distributed (networked) vs local autonomic system

A3 off-the-shelf autonomic development platform (such as IBM's ACT)

T1 relational databases for knowledge management

T2 NoSQL database (object oriented or graph model) for knowledge management

T3 formal models (e.g., temporal logic) for knowledge representation

S1 inclusion vs exclusion of plans and policies from the main knowledge repository

S2 inclusion vs exclusion of system states in the knowledge repository

S3 multifaceted vs <subject, predicate, object> property representation.

4.5.5 Quality attributes and usage scenarios

The following quality attributes[2] were elicited, prioritised (the list below is presented in ascending order of priority) and illustrated with practical scenarios.

Proactivity

Systems with the proposed architecture should make and implement decisions without human intervention. In the SAR problem domain, font size and brightness should be adjusted according to changes in the monitored variables without the presenter's explicit input. In the CDS context, heterogeneous information should be collated, interpreted and displayed in the absence of human supervision.

Usage scenarios. In the SAR problem domain, when the audience focus drops, the font and brightness are increased without disturbing the presenter. In the CDS context, users are exploring and tagging career related resources, while their annotations are captured by a sensor and integrated in the underlying knowledge base without interrupting their browsing.

Adaptive Transparency

In some application spaces, the knowledge base should be hidden from the end user to minimise disruption. In others, allowing user access to the knowledge stored by the autonomic manager may prove beneficial, especially when understanding the underlying data is vital to the type of service provided.

Usage scenarios. Presenters deliver their talk to the audience without being aware of the autonomic manager's knowledge about battery level and audience focus, as that would only distract them from their task. Career advice seekers, on the other hand, benefit from having access to data about various professions and how they relate to each other, jobs, relevant university courses, etc., as that provides them with a perspective of the field.

Realtime Operation

The systems underpinned by the proposed architecture must respond to change without significant lags. The presented document will be re-rendered as soon as a relevant trigger (e.g., a focus or battery charge drop) has been detected, with a maximum delay of the order of seconds. Although some career knowledge base maintenance may be performed offline (for instance, complete consistency checks usually take place once a day), data must be collected as it is published on job sites or annotated by career resource explorers.

Usage scenarios. During the presentation of the document, the battery level reaches a critically low level, thus, regardless of the level of focus, the system will reduce brightness to prevent power loss, before the battery is completely depleted. People doing online research about careers annotate a webpage while a new job is advertised on `indeed.co.uk` - both streams of data are captured by the appropriate sensors and included in the knowledge base as they become available.

---

[2]ATAM does not offer a clear distinction between business drivers and the quality attributes. For instance, in the method's description retrieved from `http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm`, availability and security are listed as both business drivers and quality attributes. Here, we assume that the latter reflect the former, only at a higher level of granularity.

Structural Simplicity

No additional level of complexity (at hardware or software levels) should be necessary to allow the system to meet a satisfactory standard of quality. This is of paramount importance, because simplicity supports flexibility, making the architecture adaptable to various problem domains.

Usage scenarios. The speaker does not need to plug extra hardware into the computer that the presentation is running on, nor execute any additional algorithms in order for the autonomic manager to be able to operate. Career information management (by the system) and exploration (by the user) does not require a complicated "translation" layer to turn the language of the user into one interpretable by the machine. Besides a web server configuration that is appropriate for the expected user load, the manager operating in the CDS domain does not require additional hardware to run on.

4.5.6 Architectural Approaches' Analysis

The alternative realisations (4.5.4) of the architecture were analysed with respect to the identified quality attributes. The resulting tradeoffs (that is, the quality attributes improved by a certain architecture as opposed to the ones damaged by it) were evaluated and a decision was made either in favour or against each architecture candidate.

A1 A hierarchical autonomic system (comprising several managers on different levels of abstraction, controlled by hierarchical superiors) would improve the coordination of tasks throughout the system, potentially enabling the recognition of a wider range of symptoms and a more accurate response (increased proactivity). However, that would also damage realtime operation given the lags implied by the communication between managers on different layers of the hierarchy. Since the latter quality attribute takes priority over the former, a flat architecture was adopted.

A2 A distributed autonomic system would help process large volumes of information faster (beneficial in terms of proactivity and realtime operation) yet the additional hardware and software inherent to a networked design would compromise simplicity. Thus, the architecture will contain one autonomic manager (as opposed to several spread across a grid), yet, the underpinning algorithms will be modular (configured as services) and capable of running on a PC as well as on a web server.

A3 A third party autonomic development platform would most likely improve proactivity (according to the way professional frameworks, such as ACT or ABLE, are advertised). At the same time, "one-size-fits-all" solutions tend to be heavyweight and difficult to configure, which damages the simplicity and adaptive transparency quality attributes (at the time of writing, ACT does not allow the exposure of the underlying knowledge base to the user). Consequently, the proposed architecture will be built from modular tools, that are flexible both in operation as well as in the way they can be assembled together.

T1-T3 Relational databases provide no native support for hierarchical knowledge (crucial when modelling related careers) and do not facilitate learning (only explicitly asserted facts are considered to be true). Therefore, this sort of system would have a limited capability of making decisions without being prompted (proactivity would be low). NoSQL or a formal model prove more flexible, yet would require a bespoke inference engine to support learning (damages simplicity). Also,

a supplementary translation layer would be necessary to allow non-specialist users to understand knowledge expressed in a mathematical/logical formalism. The final decision was in favour of ontologies, as they are equipped with embedded inference engines (reasoners) and lend themselves well to intuitive visualisation techniques, by exploiting the graph-like structure of RDF.

S1 Storing plans and policies in the ontology would allow the system to reason on these two components, thus increasing the accuracy of responses to complex triggers from the managed resource. At the same time, the additional reasoning complexity would damage realtime operation and simplicity. Hence, in the proposed architecture, the two components are stored separately from the main knowledge repository.

S2 The availability of system states in the knowledge base would improve the efficiency of autonomic tasks such as analysis and planning (thus increasing proactivity and improving realtime operation). Yet a complete state model of the managed resource is not always possible to extract, not to mention the ensuing increase in the size of the knowledge base (hence decreasing proactivity and damaging realtime operation). Since this design alternative seems to be both beneficial and detrimental relative to the same quality attributes, the final conclusion was to include a state model in the knowledge repository, yet instantiate it only when needed in the context of a given application (e.g., modelling the states of the managed resource in the SAR domain is straightforward, whereas the same task in the context of CDS is not computationally feasible).

S3 The inclusion of multi-faceted properties yields the same discussion as in the case of system states. Including them would allow for a more accurate representation of the managed resource with benefits in terms of proactivity, and, at the same time, would add a layer of complexity to the ontology, damaging the same quality attribute. As previously, the final decision was to provide a mechanism for multi-faceted property formulation, with an optional practical realisation.

4.5.7 Additional Usage Scenarios

A prototype implementing the proposed architecture in the CDS problem domain was piloted during several advisory group meetings, where Good Careers Guide employees and Aston University students experimented with the platform and gave feedback. The additional usage scenarios that were formulated with this opportunity prompted minor operational changes (e.g, while exploring the knowledge base in the form of a graph, the edges should not be labelled with the type of relationship they represent, as that would clutter the display). However, those were accommodated without any modifications to the architecture agreed upon in the previous ATAM step.

4.5.8 Threats

During the process of applying ATAM to analyse the KAS architecture in the CDS domain, several risk factors were identified.

- Stakeholder bias. The domain expert providing the legacy knowledge base (that the ontology is extracted from) has a clear and comprehensive view of the careers domain, stemming from years of experience in the field. However, the way the initial pool of information is structured (for instance,

the fact that there are only three types of relationships between career fields: inheritance, sibling-to-sibling and synonymy) may, at least in the early stages of using the system, bias the perception of the relevant community. Non-specialist curators may be tempted to think that the three existing types of properties are the only options available, which may stall the development of the ontology.

- Community speciation. As individual curators become aware of and even rely on each other's contributions to the ontology, groups may form that are driven by an interest in a specific sub-domain of the career field, for instance, that of science. If the edits suggested by these groups are sound, their influence may grow to the detriment of other fractions of the community, interested in non-scientific professions, that are either less efficiently coordinated or less reliable in their ontology editing activities. This may lead to some parts of the ontology being better curated than others, resulting in an uneven level of decision support service for the average job seeker.

- Industrial interest. The success of the system relies to an important extend on the interest of employers. This interest will most likely be expressed in the form of ideal candidate ontologies that job applicants can compare against their own to measure their suitability for a particular role. This feature can only be available if sufficiently important job providers are attracted by the visibility that the CDS system may offer.

All these points will be considered in the future development of the KAS architecture.

Autonomic Systems with State-featuring Ontologies

This is the description and analysis of a bespoke autonomic system built in compliance with the KAS architectural template by using (and extending) the KAS tools and by implementing the KAS methodology. The system's autonomic manager controls document rendering in response to changing environmental conditions, a case study designed to show that KAS is not just an abstract theoretical framework, but one with applicability in practical domains. For exposition continuity, the KAS tools used in this case study are described alongside the architectural components they support and not in a separate section as done in the previous chapter.

5.1 Application Domain

The **S**elf-**A**daptive Document **R**endering (SAR) problem consists in automatically modifying the way an electronic document is being displayed in response to changes in the environment (namely the audience and the machine used for the presentation). In this context, the electronic document's display medium can be identified as the *managed resource* (Fig. 5.1), as the document itself is not modified.

Specifically, the way the document is being rendered adapts to:

M1 the level of audience focus - extrapolated, for instance, from the exposed eye surface (detected by a camera) of the people in the presentation room

M2 the time of day - provided by the system clock

M3 the level of energy consumption - namely, remaining battery life.

In response to to changes in the factors above, the document display may change with respect to:

E1 font size - that may be increased to make document contents easier to see

E2 screen brightness - that may be increased (for the same reason as above) or decreased (to preserve battery levels).

In its classic form, document rendering adaptation is a manual process, that is to say that the human operator/presenter observes the battery levels and the audience's focus and explicitly changes font sizes and screen brightness accordingly. Deploying a KAS instance to automate these tasks would allow the speaker to focus exclusively on the contents of the presentation. This addresses the core requirements of the SAR application domain in the following ways (relevant to the second research question in 1.2, specifically objectives O1.3 and O2.2):

- The document must be rendered in realtime, therefore the underlying autonomic manager needs to sense the changes in the environment and respond accordingly in a limited time window. Hence, the algorithms supporting MAPE components are implemented in a fast executing environment (Java) and configured in a way that ensures a sensible order of computational complexity.

- The manager is required to ascertain that the changes to the way the document is rendered have a positive effect. In other words, questions need to be answered to determine, for instance, whether decreasing display brightness successfully preserved battery levels. Since KAS employs an ontology to store the system's knowledge, this task can be delegated to the semantic query answering service.

- In order to improve its performance, the autonomic manager should be able to learn from its experience. Specifically, a plan that successfully improved the utility of the managed resource's state in the past should be stored and reused is necessary. The plan algorithm included in the KAS framework supports this feature by maintaining a bank of known effective plans.
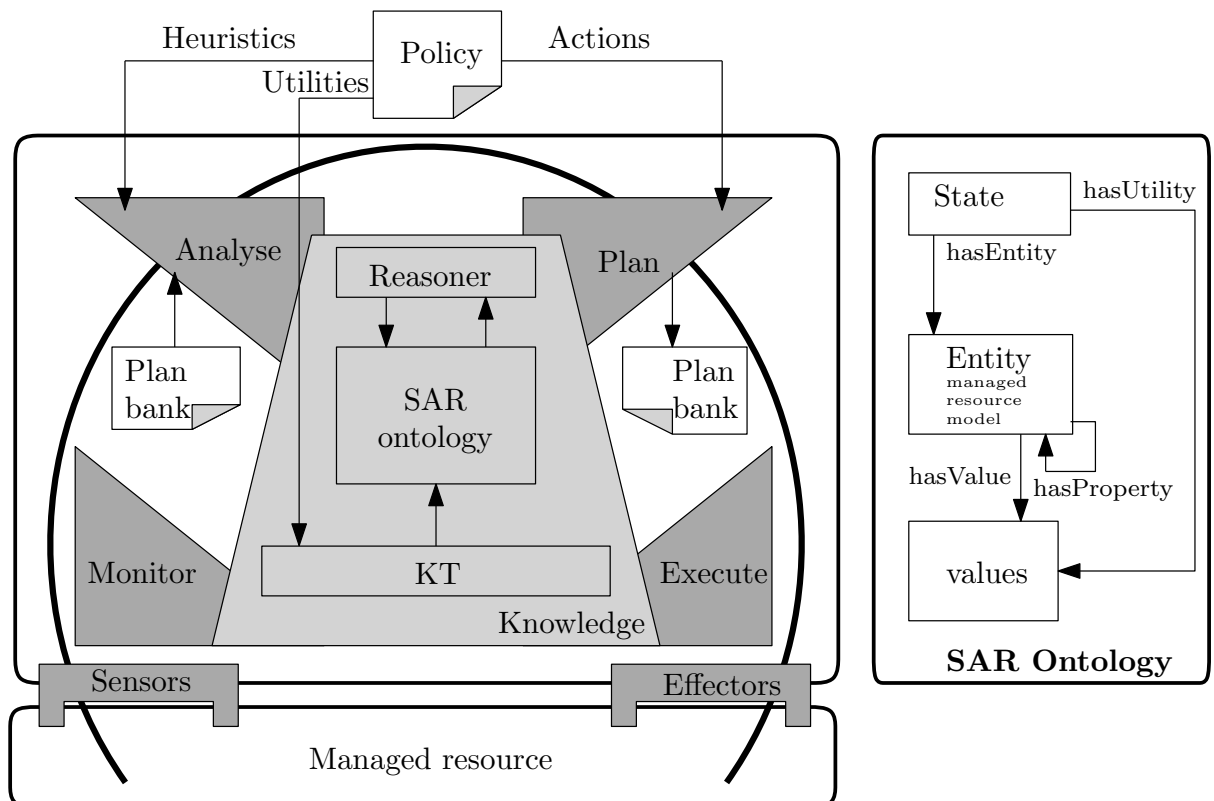
5.2 Architecture Description

Fig. 5.1: KAS with state-featuring ontology - component view

5.2.1   SAR Ontology and KT

The SAR ontology engineering process may be partially mapped against most state of the art ontology learning (OL) methodologies. For instance, the enrichment, consistency resolution and evaluation OL subtasks suggested by Petasis et al. [143] were performed during SAR ontology construction by inferring new domain knowledge, eliminating cycles as well as logical contradictions and measuring the efficiency of the ontology design, respectively. However, the most intuitive methodology to describe the SAR ontology engineering process is NeOn [168]. The way that the first NeOn scenario (for OL from scratch) is implemented in the SAR problem domain is described in the following.

Ontology requirements specification

- Purpose definition. The goal of KAS when deployed to solve the SAR problem is to "maximise audience focus while minimising power consumption". The purpose of the ontology is to enable and support the realisation of that goal. To achieve it, the system should be able to evaluate *audience focus* and *battery consumption*, which prompts the need for modelling those two concepts in the ontology.

- Scope definition. Given the purpose definition, it follows that the ontology should represent all possible states of the SAR environment, namely all combinations of legal values for **M1**, **M2** and **M3**. This is done by automatically asserting a hierarchy of `State` concepts, one of which is shown in Fig. 5.2[1].

```
Class: <SARonto#State1> EquivalentTo:
  <SARonto#hasTimeOfDay> some <SARonto#EarlyMorning>
  and (<SARonto#hasFocus> some <SARonto#MediumFocus>)
  and (<SARonto#hasBattery> some <SARonto#HighBattery>)
  and (<SARonto#hasUtility> some xsd:integer)
```

Fig. 5.2: `State1` concept definition

Classes and properties are prefixed with the ontology URI - in this case, `SAROnto#` - to make every name unique (for simplicity, since SAR uses only one ontology, the URI will be omitted in text). `EquivalentTo` shows that `State1` is a defined class (as opposed to a primitive class), namely that the four conditions to follow are both necessary and sufficient. Defining classes this way is compulsory, as semantic reasoning can not be performed on primitive classes. The actual definition of class `State1`, namely the conjunction of the four necessary and sufficient conditions, makes use of object properties `hasTimeOfDay`, `hasFocus` and `hasBattery` as well as data property `hasUtility`. The first three properties link `State1` to specific values of **M1** (`MediumFocus`), **M2** (`EarlyMorning`) and **M3** (`HighBattery`). These are discrete classes of values for the three continuous inputs and will have to be properly defined in the ontology conceptualisation step. The

---

[1]All ontology excerpts presented from here onwards are formulated in Manchester OWL syntax (`https://www.w3.org/TR/owl2-manchester-syntax/`), a frame-based format that is more compact and easier to understand than axiom-based ones, such as XML, Turtle or most forms of predicate logic.

fourth property associates a numeric utility (prescribed in the policy document) to each possible system state.

- Ontology language selection. The SAR ontology is developed in OWL 2 (the latest, industry standard version of the Web Ontology Language). This is the most recent W3C ontology language recommendation[2] for the Semantic Web. OWL [10, 86] is built on top of the RDF schema but is more expressive, allowing more complex descriptions of domain knowledge. Two examples of OWL semantics that are not available in RDF Schema have already been illustrated in the definition of class `State1` above, namely using properties to define restrictions for individual subclasses (RDF would have allowed restrictions for the entire `State` hierarchy only) and defining a class as a boolean combination (conjunction) of other classes (all four restrictions in the `State1` definition are actually anonymous classes). For a full list of OWL features that are not available in RDF Schema, see [10]. The final reason for choosing OWL 2 is the availability of convenience tools, such as Protege [82], a powerful editor for OWL ontologies, and the OWL API[3], a comprehensive library allowing the manipulation of OWL ontologies from Java applications.

- Intended users definition. The SAR ontology is meant to store the autonomic manager's knowledge base. There is no direct interaction between the human users of this KAS instance and the ontology, therefore no visual interface is in place. However, the ontology content is processed by the Java modules underpinning the MAPE components, a connection facilitated by the OWL API.

- Requirements definition. The SAR ontology stores information about the state of the environment (battery life, audience focus and system time). However, the `State` hierarchy concepts cannot be created manually, since the ontology is not exposed to the end user of the KAS instance, the only human stakeholder with access to state information. Thus, the first requirement for the SAR ontology is that it is automatically created (the tool responsible with automated OL is discussed in the ontology implementation section). Moreover, the autonomic manager drawing knowledge from the ontology needs to operate in realtime, implying a swift identification of the current state of the system (analysis), the selection the appropriate plan and the execution of the necessary operations with minimum delay in order to ensure a seamless presentation. To enable this, the second requirement of the SAR ontology is compactness: a lightweight knowledge base, unburdened by concepts that are not reasoned on. Consequently, plans and policies are stored in the Java engine powering the MAPE control loop and not in the ontology.

Ontology Life Cycle Configuration

The SAR ontology goes through three main stages:

- Creation. SAR ontology learning is performed automatically and implies two sub-stages: discretisation, namely asserting classes to represent categories of values for the continuous **M1**, **M2** and **M3** inputs, and `State` hierarchy generation, containing concepts such as `State1` above (the utilities associated to each state are taken from the from the policy document).

---

[2]https://www.w3.org/TR/owl2-overview/

[3]http://owlapi.sourceforge.net/

- Enrichment. At given intervals of time (depending on the monitoring frequency), **M1**, **M2** and **M3** values are read from the environment. The reasoner will automatically infer (see 5.2.2 for an example) which discrete class those values belong to, thus allowing the identification of the current system state (necessary during analysis).

- Verification. After reading each new set of **M1**, **M2** and **M3** values, the reasoner verifies that the values are in range. This prevents sensor malfunctions, as hour 26 or battery level -10 would render the `State` concept unsatisfiable (this is illustrated in 5.2.2).

Ontology Conceptualisation

The SAR ontology in Fig. 5.1 instantiates the Onto++ template in Fig. 4.1. Since there are no multifaceted properties (e.g., no weights), there is no need for reification, hence the `Link` hierarchy has been omitted. The remaining two hierarchies, `Entity` and `State`, as well as the properties connecting them are depicted in more detail in Fig. 5.3 and described in the following.
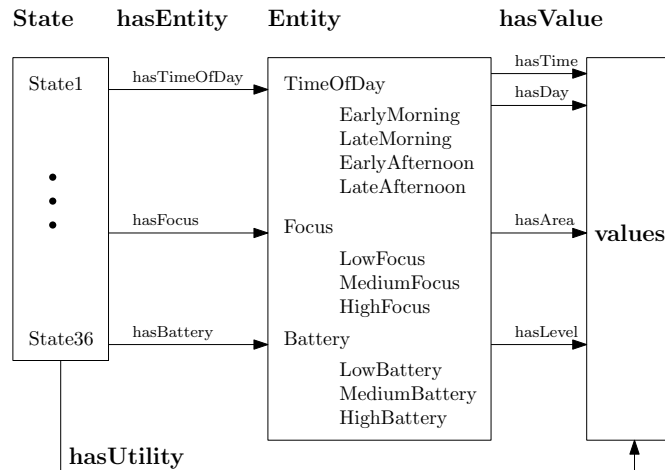

Fig. 5.3: The SAR Ontology in more detail - concept hierarchies are presented as indented lists

- The `Entity` hierarchy The `Entity` class is directly subsumed by `TimeOfDay` - modelling **M2**, `Focus` - representing **M1**, and `Battery`, describing **M3**. The time of day falls in one of four categories (details about defining these categories are provided in the ontology implementation section): `EarlyMorning` (Fig. 5.4), `LateMorning`, `EarlyAfternoon` and `LateAfternoon`.

  Concept `TimeOfDay` (Fig. 5.4) is defined by two value properties: `hasDate` links the concept to a month of the year, namely an integer between 1 and 12, and `hasTime` relates it to an hour of the (working) day, an integer between 9 and 17. Concept `EarlyMorning` refers to a more restrictive set of times, namely between 9 am and 10 am in the summer (that is, between the third and ninth month of the year) or between 9 am and 11 am in the rest of the year. This provides a flexible definition for what constitutes "early morning", in that it allows for an extra hour in the winter. The other `TimeOfDay` subclasses are defined in a similar fashion to `EarlyMorning`. `LateMorning` starts at 10 am in the summer / 11 am in the winter and finishes at noon. `EarlyAfternoon` starts at noon and ends at 3 pm in the summer / 2 pm in the winter, which is also when `LateAfternoon` starts. `LateAfternoon` ends at 5 pm.

```
Class: <SARonto#TimeOfDay> EquivalentTo:
  <SARonto#Entity>
  and (<SARonto#hasDate> some xsd:integer[>0 , <=12])
  and (<SARonto#hasTime> some xsd:integer[>=9 , <=17])
```

```
Class: <SARonto#EarlyMorning> EquivalentTo:
  <SARonto#Entity>
  and ((((<SARonto#hasDate> some xsd:integer[>0 , <=3])
      and (<SARonto#hasTime> some xsd:integer[>=9 , <11]))
    or ((<SARonto#hasDate> some xsd:integer[>3 , <=10])
      and (<SARonto#hasTime> some xsd:integer[>=9 , <10]))
    or ((<SARonto#hasDate> some xsd:integer[>10 , <=12])
      and (<SARonto#hasTime> some xsd:integer[>=9 , <11))))
```

Fig. 5.4: `TimeOfDay` and `EarlyMorning` concept definitions

```
Class: <SARonto#State> EquivalentTo:
  <SARonto#hasTimeOfDay> some <SARonto#TimeOfDay>
  and (<SARonto#hasFocus> some <SARonto#Focus>)
  and (<SARonto#hasBattery> some <SARonto#Battery>)
  and (<SARonto#hasUtility> some xsd:integer)
```

Fig. 5.5: The `State` hierarchy root

Note that, given the discretisation threshold values (10 am/11 am for mornings and 2 pm/3 pm for afternoons) the `TimeOfDay` hierarchy can be easily built automatically (the algorithm to do that is described in the ontology implementation section). This is true for the `Focus` and `Battery` hierarchies as well. The discretisation threshold that separates `LowFocus` from `MediumFocus` is 10 units of exposed eye area, whereas the separation between `MediumFocus` and `HighFocus` is done at 20 units. Similarly, the level of charging goes from `LowBattery` to `MediumBattery` once over 25%, and from `MediumBattery` to `HighBattery` once over 75%[4].

- The `State` hierarchy. Classes `State1` (Fig. 5.2) up to `State36` each represent a combination of discrete values for the three monitored inputs. In addition, each state class has an associated *utility* (provided in the policy document), a number that can be used to evaluate the closeness to the goal state. All state concepts are automatically generated and subsume `State` (Fig. 5.5).

- The `hasEntity`, `hasProperty` and `hasValue` properties. `State` hierarchy concepts are related to concepts from the `Entity` hierarchy, namely `TimeOfDay`, `Focus` and `Battery` via *object properties*, specifically `hasTimeOfDay`, `hasFocus` and `hasBattery`, respectively. Relative to 5.1, these properties are instances of the generic `hasEntity` root property. *Data properties* `hasTime`, `hasDay`, `hasArea`, `hasLevel` and `hasUtility` connect `Entity` and `State` subconcepts to numerical values. They are instances of the generic `hasValue` property in Fig. 5.1 and 5.3. The only top level property in the general view of the SAR ontology (Fig. 5.1) that does not appear in

---

[4]The formal definitions for the `Focus` and `Battery` hierarchies are very similar to the `TimeOfDay` one and have been omitted from the main text

the detailed view (Fig. 5.3) is `hasProperty`. In the SAR scenario, `Entity` concepts are connected to each other by means of inheritance only, a type of relationsip that OWL asserts silently (there is no explicit `hasParent` property).

Ontology formalisation and implementation

In the original NeOn methodology [168], the output of this OL phase is a "computable model" of the ontology. This is generated automatically within the KAS framework, by the Knowledge Translator (KT) (described in 4.3.2), an OL algorithm from the KAS tool set that is portable across application domains. In order to create the SAR domain ontology, the KT algorithm receives a specific set of input values.

Input `inputFile`. There is no legacy description of the SAR system, therefore this parameter is null. The `Entity` hierarchy will be created via discretisation and the utilities associated to each state of the system will be extracted from the policy document.

Input `isDiscrete`. Since SAR monitors three continuous signals (**M1**, **M2** and **M3** introduced in 5.1), the value of this input is `false`.

Input `thresholds`. The threshold values may be directly used for discretisation or combined in more complex boolean expressions. Consider the thresholds for the **M1**, **M2** and **M3** monitored inputs, given in Table 5.1. The concepts asserted for **M1** are straightforward: `LowFocus` corresponds

Table 5.1: KT `thresholds` values - an example

	M1	0	30	60	100			
			time				date	
	9	10	12	15	17		3	10
M2	9	11	12	14	17		1	3
	9	11	12	14	17		10	12
	M3	0	25	75	100			

to less that 30% exposed eye surface, `MediumFocus` to the 30% - 60% interval and `HighFocus` to over 60%. Similarly, there will be three categories of values to represent **M3**: `LowBattery` - under 25%, `MediumBattery` - between 25% and 75% and, respectively, `HighBattery` - over 75% battery charge. In the case of **M2**, the threshold values on the same row but in different boxes are combined using the `and` logical operator, whereas values on separate rows are aggregated using `or`, ultimately leading to four concepts: `EarlyMorning`, defined in Fig. 5.4, and similarly constructed `LateMorning`, `EarlyAfternoon` and `LateAfternoon`.

Output o. The SAR ontology returned by KT will contain the `Entity` and `State` concept hierarchies, along with their associated properties, as shown in Fig. 5.3. The construction of output o follows the KT algorithm logic explained in 4.3.2 (except for the stage concerning `Link` hierarchy creation, which is omitted since there are no multi-faceted properties in the SAR problem domain).

5.2.2 Reasoner

Out of the set of reasoner features exploited by KAS (described in 4.3.1), the SAR problem domain makes use of two:

Subsumption. This is exploited by KT during discretisation and by the MAPE monitor component. The KT algorithm asserts a direct `Entity` subconcept for every monitored input (in the case of SAR, they are `TimeOfDay`, `Focus` and `Battery`, as shown in Fig. 5.3). To illustrate subsumption, let us analyse the `TimeOfDay` hierarchy, obtained by discretising the **M2** monitored input and comprising concepts `EarlyMorning`, `LateMorning`, `EarlyAfternoon` and `LateAfternoon`. Initially, these last four concepts are asserted under `Entity`, however, the reasoner is able to infer that they are, logically, subconcepts of `TimeOfDay`. Indeed, given the formal definitions of `TimeOfDay` and `EarlyMorning` (Fig. 5.4), it is straihforward to conclude that the latter refers to a subcategory of `TimeOfDay` instances, namely certain hours of the day in certain months of the year. The process is graphically illustrated in Fig 5.6, where `earlyMorning` is asserted as a direct subclass of `Entity` but inferred as a direct subclass of `TimeOfDay`.
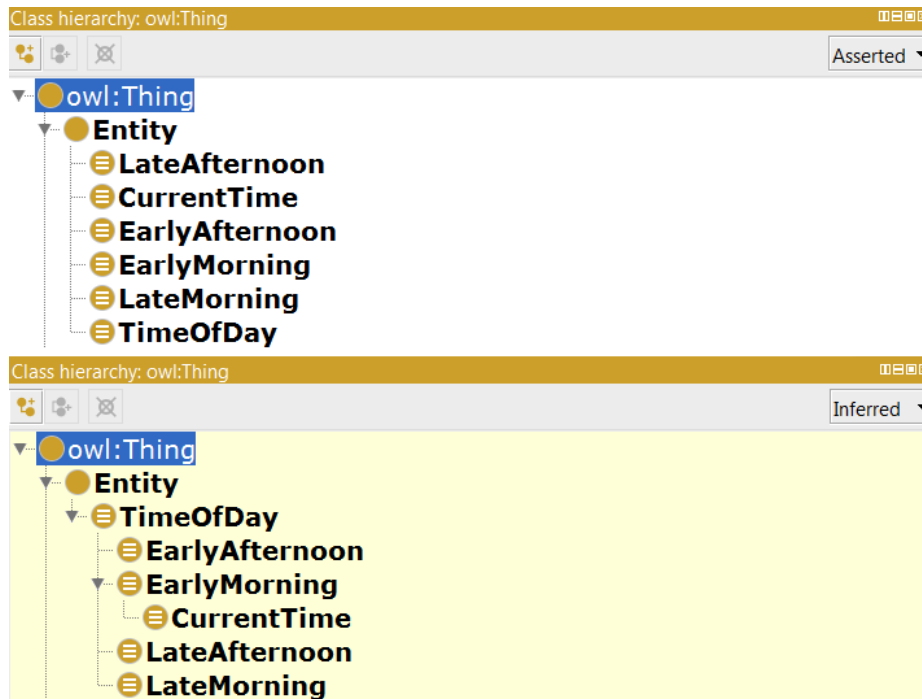

Fig. 5.6: The `Entity` hierarchy before (Asserted) and after (Inferred) subsumption (Protege ontology view, where hierarchies are displayed as indented lists)

The monitor component exploits subsumption while asserting ontology concepts for newly read input values. Let us continue relating to **M2** and assume the current time is 10 am in November. Even though `CurrentTime` (Fig. 5.7) is initially asserted (by KT) as a direct subclass of `Entity`, the reasoner will infer its appropriate place in the hierarchy, namely under `EarlyMorning` (Fig. 5.6). This has an important impact on KAS's capacity of processing knowledge, as it allows the system to "understand", without being explicitly "told", that 10 am on a November day qualifies as early morning, whereas in July, the same time belongs to the late hours of the morning.

Sensor data verification. Let us assume that the sensor reading **M2** values is defective and returns hour -1 (the month is irrelevant for this example). The `currentTime` concept in Fig. 5.7 is asserted with filler [>=-1 <= -1] for property `hasTime`, which does not match any of the `TimeOfDay` class definitions. Consequently, `CurrentTime` will be inferred under `OWLThing` (the base of the OWL concept hierarchy). The current state of the system will therefore violate the definition of the `State` (Fig. 5.5)

```
Class: <SARonto#CurrentTime> EquivalentTo:
  <SARonto#Entity>
  and (<SARonto#hasTime> some xsd:integer[>= 10 , <= 10])
  and (<SARonto#hasDate> some xsd:integer[>= 11 , <= 11])
```

Fig. 5.7: The `CurrentTime` concept definition

class (since the `hasTimeOfDay` filler is `OWLThing` instead of a subclass of `TimeOfDay`), thus making the `State` concept unsatisfiable. Sensor data validation is not an implicit feature of the resoner, instead it is facilitated by the structure of the ontology (provisioned with a `State` hierarchy with the appropriate property restrictions).

5.2.3 Policy

The SAR policy document contains all three sections described in 4.2.2.

Utilities. The domain expert (and author of the policy document) may express preference towards SAR system states by assigning numeric utilities to each of them, as shown in Table 5.2. Monitored input **M2** is not taken into account, as the time of day when the document is shown to the audience does not directly influence the quality of the presentation (it does so, indirectly, as illustrated by the policy heuristics). Hence, the same utility is assigned to a group of four states to account for all discrete categories of **M2** values (early/late morning/afternoon). The goal states are **s33-s36**, corresponding to high audience focus and high levels of battery charge, which are assigned the maximum utility. Note that Table 5.2 is not the only way to associate utilities to states, as, in some other practical scenario, it may be the case that states **s5-s8** are more valuable than states **s13-s16** and their utilities should be swapped.

Table 5.2: SAR policy document - state utilities

State	M1 (focus)	M3 (battery)	Utility
s1 - s4	low	low	1
s5 - s8	low	medium	2
s9 - s12	low	high	3
s13 - s16	medium	low	4
s17 - s20	medium	medium	5
s21 - s24	medium	high	6
s25 - s28	high	low	7
s29 - s32	high	medium	8
s33 - s36	high	high	9

Actions. Given the two effectors presented in 5.1, the autonomic manager may alter the managed resource by carrying out the actions in Table 5.3. The set of actions was compiled under the following assumptions:

- decreasing the font size will not enhance audience focus nor impact battery charge levels, therefore this operation is not included in any of the actions

- increasing the font size will likely increase audience focus without impacting battery charge levels

- increasing brightness may increase audience focus at the cost of accelerating battery depletion

- decreasing brightness will likely decrease audience focus yet slow down the battery depletion speed.

Admittedly, the assumptions may change from one domain expert to another (it is, after all, plausible to expect a smaller font size and a dim display to entice the audience to focus more). However, the list given above is not intended to cover all possible scenarios, but merely to illustrate this section of the policy document and to highlight the fact that the two components of the goal (**s9** in Fig. 5.2) are conflicting, that is, aiming for high audience focus will most likely reduce battery charge levels.

Table 5.3: SAR policy document - plan actions

Action	E1 (font)	E2 (brightness)
a1	increase	increase
a2	increase	maintain
a3	increase	decrease
a4	maintain	increase
a5	maintain	maintain
a6	maintain	decrease

Heuristics. This section of the policy document contains high level knowledge used to simplify the analysis and planning stages of the MAPE loop. To illustrate, let us consider the two heuristics given in Table 5.4 (in ECA format).

Table 5.4: SAR policy document - heuristics

(a) heuristic 1

When focus is low AND	*event*
the time is late morning OR	*condition 1*
the time is early afternoon THEN	*condition 2*
exclude **a2**, **a3**, **a5** and **a6**.	*action*

(b) heuristic 2

When battery is low THEN	*event*
exclude **a1** and **a4**.	*action*

When the level of light in the environment is high (in the late morning and early afternoon), that may interfere with the display, therefore decreasing or maintaining the brightness will compromise the visibility of the document. Heuristic 1 eliminates those actions from the list of candidates to consider when formulating plans. Also, when the battery is low, increasing brightness will only drain it faster. To preserve battery levels, heuristic 2 eliminates all actions implying an increase in screen brightness. Note that, should both heuristics be simultaneously activated (in case audience focus is low and so is the battery charge level), there would be no actions left to consider during planning. To prevent the system from going idle, the heuristics are prioritised: in the provided example, heuristic 2 takes precedence over heuristic one.

5.2.4 Monitor, Execute and the Managed Resource

The **monitor** samples inputs **M1** (audience focus), **M2** (time of day) and **M3** (battery charge level). In the KAS implementation for SAR, the data for **M1** is simulated, whereas **M2** and **M3** values are read in from the actual host PC via APIs. In terms of output, the monitor component asserts one ontology concept for each of the inputs, specifically `CurrentTime`, `CurrentFocus` and `CurrentBattery`, and links them, via the appropriate properties, to the values read in by the sensors. The definitions provided in Fig. 5.7 (`currentTime`) and Fig. 5.8 (`currentFocus` and `currentBattery`) assume that the current time is 10 am in November, that the average exposed eye area of the audience is 45% and that the battery charge is 30%. The ontology concepts asserted during monitoring will be used during analysis to determine the current state of the system.

```
Class: <SARonto#CurrentFocus> EquivalentTo:
  <SARonto#Entity>
  and (<SARonto#hasArea> some xsd:integer[>= 45 , <= 45])
```

```
Class: <SARonto#CurrentBattery> EquivalentTo:
  <SARonto#Entity>
  and (<SARonto#hasLevel> some xsd:integer[>= 30 , <= 30])
```

Fig. 5.8: The `CurrentFocus` and `CurrentBattery` concept definitions

The **execute** component implements APIs for controlling effectors E1 and E2. It will carry out the sequence of actions developed during planning.

The **managed resource** is represented by the document rendering media (e.g., Adobe Acrobat) and the computer hosting the presentation[5]. **Sensors** will read **M2** and **M3** values from the host computer, whereas **effectors** will control **E1**, a parameter of the rendering system, and **E2**, a feature of the host computer. The sensor reading **M1** does not operate on the managed resource per se, but rather on its environment (where the audience is).

5.2.5 Analyse

The analyse algorithm is a KAS tool described in Table 4.3. The specific implementation of this tool within the SAR problem domain is explained in the following by discussing each of the algorithm's inputs and outputs, namely their initial values and the way they are processed throughout analysis.

Input o. During the monitoring stage, the reasoner will have subsumed the sensor data from the **M1**, **M2** and **M3** signals in their rightful categories. Thus, `CurrentTime` (Fig. 5.7) is inferred to be a subclass of `EarlyMorning`, whereas `CurrentFocus` and `CurrentBattery` (Fig. 5.8) are placed under `MediumFocus` and `MediumBattery`, respectively. Given this setup of the ontology, method `findCrtState` (line 1 in Table 4.3) will compare the three `Entity` subclasses above against the fillers of all 36 states and identify the matching one, shown in Fig. 5.9.

---

[5]The actual electronic document is not considered to be part of the managed resource as its content on the disk does not change.

```
Class: <SARonto#State17> EquivalentTo:
  <SARonto#hasTimeOfDay> some <SARonto#EarlyMorning>
  and (<SARonto#hasFocus> some <SARonto#MediumFocus>)
  and (<SARonto#hasBattery> some <SARonto#MediumBattery>)
  and (<SARonto#hasUtility> some xsd:integer[>=5, <=5])
```

Fig. 5.9: The `State17` concept definition

Input `policy`. There are two heurisics (Table 5.4) in the policy document, neither of which refer to `State17`. Therefore the set of candidate plan actions in Table 5.3 cannot be simplified.

Output `u`. This is the utility associated to the current system state, `State17` in Fig. 5.9, namely 5.

Output `actions`. The full set of actions in the policy document is returned by the analyse algorithm to be considered during planning.

5.2.6 Plan and the Plan Bank

The planning stage of the MAPE control loop, as implemented by KAS, is described in Table 4.4. In the SAR problem domain, the general plan algorithm receives specific inputs, discussed in the following.

Input `u`. Assuming that the current system state is `State17` (Fig. 5.9), the associated utility, `u`, will be 5.

Input `actions`. Since the analyse algorithm was not able to eliminate any of the actions defined in the policy document, the `actions` set passed into the plan algorithm will contain all six actions in Table 5.3.

Input and output `p`. The first time the plan algorithm is run, `p` will be empty. It will be populated with either a sequence of actions that is known to be effective in increasing the system's state utility (namely, a plan from the bank) or, if no such sequence exists for the given initial utility, with a randomly selected action from the `actions` set.

Input `planBank`. Initially, the plan bank is also empty. After executing the plan algorithm in a loop, the current plan `p` will be saved in the bank, if the utility of the system state reached after executing the sequence of actions in `p` is higher than the initial utility `u`. Relative to the running example, a plan will be saved in the bank if it drives the system to a state with a utility greater than 5.

5.3 Implementation Scenario

One step of the generic KAS methodology in Table 4.5 is illustrated on the running example used throughout this chapter. This will show how the KAS architecture can be instantiated and run in the SAR problem domain by following the algorithm logic in 4.4. The process is captured by the flow diagram in Fig. 5.10.

5.3.1 Setup and ontology learning

Policy formulation. KAS deployment in the SAR problem domain starts with defining state *utilities*, candidate plan *actions* and *heuristics*, a task performed by the domain expert. System states
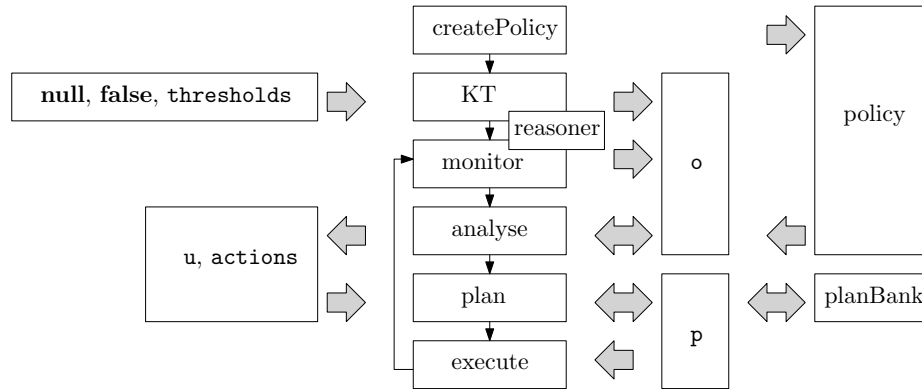
Fig. 5.10: The KAS methodology instantiated in the SAR problem domain - the architecture tools are displayed in the centre and connected via linear arrows to show the sequence in which they are executed; the tools' inputs and outputs are displayed on either side and connected via block arrows

are combinations of value ranges applicable to monitored inputs (audience focus and battery charge level), whereas plan actions are generated with respect to the controlled (output) parameters (font size and display brightness). Heuristics capitalise on the expert's previous knowledge and can be viewed as known good strategies to apply in order to drive the system into a state with a utility higher than the current one. The recommended complete layout of the SAR policy document is available in Tables 5.2 (utilities), 5.3 (actions) and 5.4 (heuristics).

Creation of the initial ontology. Since SAR is a domain with continuous monitored inputs, no legacy description (input file) is necessary. The KT algorithm will create the `Entity` hierarchy via discretisation (as explained in 5.2.1), hence the first input parameter (`inputFile`) will be null and the second (`isDiscrete`) will be `false`. The third KT input, `thresholds`, containing the values used for discretisation, is prescribed by the application developer (an example is provided in Table 5.1). The initial ontology produced by KT and illustrated in Fig. 5.3 is an instance of Onto++ from the general KAS architecture (Fig. 4.1), in that all Onto++ concept hierarchies are present in the SAR ontology, save `Link`, since this problem domain can be modelled without multi-faceted properties. Note the role of the reasoner during the discretisation process, as illustrated in 5.2.2.

5.3.2 Monitoring and analysis

Input processing. In the SAR application domain, KAS processes three inputs during the monitoring phase: the time of day and the battery levels are read from the computer (that the presentation is running on) clock and power manager, respectively, whereas the data representing the area of exposed eye (averaged over the entire audience) is simulated. At 10 am on a November day, with 30% battery charge and an average exposed eye area of 45%, the monitor will assert three concepts in the ontology, `CurrentTime` (Fig. 5.7), `CurrentFocus` and `CurrentBattery` (Fig. 5.8). After the monitor phase, the ontology in Fig. 5.3 is enriched with the three concepts above, each subsumed by the reasoner as shown in Fig. 5.6. Additionally, the reasoner implicitly validates sensor data as explained in 5.2.2.

Analysis. Given the inferred super-classes for the three concepts asserted in the ontology during monitoring, the analysis module will identify the current state of the system as `State17` (Fig. 5.9). The associated utility will be returned for the use of the planner. Since there are no heuristics in the policy

document adivising which actions to avoid early in the morning, when the audience focus and battery level are both in the medium range, the analyser will return the full set of candidate actions (Table 5.3) to be considered during planning.

5.3.3 Planning

Let us assume that a plan may be, at most, three actions long ($L = 3$) and illustrate the planning loop from the general KAS methodology (Table 4.5, lines 11 - 16) on the running example.

Iteration 1. The utility of `State17` (medium focus and medium battery level) is 5, thus lower than that of the goal state (Table 5.2 shows that the maximum utility, 9, is associated to high focus and high battery level). Since the plan bank is initially empty, an action will be selected from the available set in Table 5.3: let us assume that is **a2** (increase font and maintain brightness). Let us also assume that executing action **a2** will successfully increase audince focus. Hence, the values of the input signals detected by the monitor are 10 am for time, 30% for battery level and, say, 70% for exposed eye surface, corresponding to concepts `EarlyMorning`, `HighFocus`, `MediumBattery`. The associated system state determined by the analyser is `State29` with utility 8 and no policy heuristics referring to it.

Iteration 2. Since `State29` does not coincide with the goal one, the algorithm continues by selecting another action from Table 5.3, for instance **a4**. Assuming that maintaining the font size and increasing the display brightness has no impact on audience focus but lowers the battery charge level to 20%, the concepts asserted during monitoring will be subsumed under `EarlyMorning`, `HighFocus` and `LowBattery`, respectively. During analysis, the system state is found to be `State25`, with utility 7 and referred to by the second heuristic (Table 5.4). Consequently, the set of candidate plan actions is reduced to {**a2**, **a3**, **a5**, **a6**}.

Iteration 3. The goal state utility has still not been reached, therefore another action (for example, **a6**) is selected from the set of candidates produced during analysis. The font size is maintained whereas the appropriate effector will decrease screen brightness, leading to no change (relative to the previous iteration) in the three monitored inputs. The analyse stage will preserve `State25` with utility 7 as the current system state, apply the second policy heuristic and return the same set of candidate plan actions as before. However, the fourth iteration will not take place since the maximum allowed number of plan actions ($L = 3$) has been reached.

5.3.4 Maintaining the plan bank

The final utility achieved by executing the previously constructed plan is 7, a higher value than the utility of the initial `State17`, namely 5. This validates the condition on line 18 in the general KAS methodlogy (Table 4.5) and leads to plan {**a2**, **a4**, **a6**} being included in the plan bank.

5.3.5 Self-management support

The KAS implementation in the SAR domain supports the realisation of the four self-management properties [101].

- **Self-optimisation.** The autonomic system's behaviour is optimised with regards to the managed resource (the document rendering media, namely, the computer monitor and the software document

manager, for example Adobe Acrobat Pro) and the autonomic control element alike. The former optimisation aspect entails adjusting document font sizes and screen brightness to account for the time of day, battery level and audience focus. The latter type of optimisation stems from the inner logic of the planning algorithm (4.3.5), where sequences of actions that have previously been successful at increasing the utility of the system state are stored in a bank and re-utilised as necessary. In other words, the MAPE-K loop exploits its own past experience, thus this facet of optimisation may also be labelled self-learning.

- **Self-protection.** In its current implementation, the KAS instance deployed in the SAR context is exposed to sensor malfunction. Should there be any data corruption with monitored input, for instance, a reported time of the day outside of the $0 - 24$ range, a battery level grater than 100 or an average exposed eye area in the negative domain, the reasoner would reject the corresponding ontology entries as shown in section 5.2.2. The autonomic element protects against its own algorithmic pitfalls: if the random action selection process used to dynamically build a candidate plan fails to reach the goal state in a reasonable number of iterations, the perceivable lag in the system's response would be unacceptable. To prevent that, there is a programmable limit in place (Table 4.5, line 11) to restrict the maximum plan length to a sensible value.

- **Self-healing.** The system is not currently equipped to handle environmental hardware issues. The future development agenda includes support for healing strategies such as switching to an alternate display when the lead monitor is malfunctioning and adjusting the other light sources in the room (close/open the blinds, dim the ceiling lights, etc.) when the main monitor's brightness controls are not responsive. This would require a more integrated control set-up (an actual "smart environment") than the one simulated here.

- **Self-configuration.** This property is relevant when the managed resource is a network (comprises several, linked components). In the SAR domain, the KAS compliant autonomic manager could be used to control several monitors in the presentation room and configure those independently. In one of the possible scenarios, where the room's energy consumption needs to be reduced while the detected level of audience focus is low, the autonomic manager would configure the array of room monitors by dimming/switching off the side ones, while increasing the brightness of the frontal display only.

5.4 Evaluation

The applicable quantitative and qualitative evaluation criteria presented in 2.1.3 and 2.2.5 are used to assess the performance of both the autonomic and semantic components of the KAS instance deployed in the SAR application domain. Some of the experiments will inherently expose information about the performance of the KAS instance for SAR, as a whole, whilst others will highlight the interaction between the two components. These aspects will be interpreted in the reflection section.

5.4.1 Autonomic Manager Evaluation

The instance of the KAS architecture deployed in the SAR application domain achieves the **qualitative indicators** (refer to 2.1.3 for detailed definitions) below.

- **Autonomicity:** level 4 (adaptive). The KAS for SAR instance pursues the high level goal of driving the managed resource to the state of maximum utility (high audience focus and full battery). It proactively compiles plans and executes them in response to changes in the monitored environment. The operational requirements preventing the KAS for SAR instance from being fully autonomic (level 5) are:

    - the domain expert has to define candidate plan actions in the policy document (heuristics and utilities would be necessary for fully autonomic systems as well),

    - the application developer needs to programme the discretisation thresholds, the final input parameter of the KT algorithm - Table 4.1 (a truly autonomic system would learn these parameters in an unsupervised way),

    - the application developer must configure the sensor and effector APIs (a level 5 autonomic system would be capable of identifying and employing the appropriate web services to perform API configuration without human involvement).

- **Architecture:** flat. There is only one autonomic manager controlling two programmable parameters (brightness and font size) of the legacy resource (the electronic document rendering software and the host computer).

- **Adaptation approach:** mixed (policy[6] and utility based). The policy document contains both utilities for each of the system states as well as heuristics (experience derived rules) to guide plan formulation and expectedly expedite the system's evolution towards the goal state.

- **Learning:** supported. The KAS architecture, as implemented in the SAR domain, provides for a plan bank, storing previously used, good quality (in the sense that their execution lead to an increase in utility) plans. They will be reconsidered during future MAPE iterations for either direct deployment or possible improvement, a mechanism that embodies the system's capacity of learning from previous experience.

- **Open:** freely available. All third party software (Java APIs for PDF management[7], battery level consultation and brightness control as well as the Protege editor[8] for OWL ontologies) used to build the KAS instance for the SAR problem domain are available under a public licence. The other KAS tools and architecture components relevant to the SAR scenario are fully disclosed (5.2).

---

[6]Movahedi et al. use term "policy" to refer to high level rules, extracted from the expertise of human specialists, meant to guide system behaviour [131]. Here, these rules are called "heuristics", whereas "policy" is endowed with a broader meaning, namely all forms of experiential knowledge (actions, state utilities as well as heuristics) that the domain expert can provide about the managed resource in addition to its structural description.

[7]http://pdfbox.apache.org/

[8]http://protege.stanford.edu/

- **Evolvable.** The proposed autonomic system is extensible and maintainable. Regardless of the number and nature of the monitored inputs, the KT algorithm will automatically create the ontological state representation accordingly[9] (provided that the appropriate thresholds are set for each monitored input and the necessary APIs are made available by the application developer). Also, the suggested MAPE infrastructure is modular, with highly cohesive algorithms implementing each phase. Thus, for example, the current version of the planner (Table 4.4) can be swapped with a new one (implementing a different action selection mechanism), given that the interface (input and output parameters configuration) stays the same.

- **Robustness.** The ontology cannot be accessed from outside the autonomic manager, therefore the `State` hierarchy is guaranteed to be complete and logically correct (given that it is automatically generated by the KT algorithm in a specific format). However, the ontology is not completely protected against sensor malfunction (the reasoner can detect sensor values that are out of range, such as hour 26 - see 5.2.2 - but cannot diagnose a system clock malfunction that tells the system it is 9 am, when, in fact, it is noon).

- **Validation.** A combination of qualitative and quantitative analysis methods are used in this chapter to evaluate the proposed KAS instance at both the architectural as well as the runtime performance levels. A mixture of simulated (camera output) and real (battery level and system time) data is considered in the next section.

Qualitative evaluation summary. Most of the qualitative indicators are positive: KAS for SAR is capable of learning from experience, has a open architecture, can be extended and maintained and is thoroughly evaluated via a mix of methods. There are two criteria where KAS for SAR can be found lacking: autonomicity (level 4 instead of 5) and robustness (medium instead of high). To address the first shortcoming, an extra component is needed to interface the monitor module with the KT algorithm. This component should be able to read online descriptions of semantic web services [184], select the ones capable of configuring the APIs for the employed sensors and execute them, thus allowing KT to receive monitored data without the application developer's help. The execute module would also use this component to automatically configure APIs for effectors. The added benefit of applying a semantic web service approach is platform independence (currently, the Java APIs that read the system time and display brightness are specific to the Windows operation system only). The second issue may be addressed by comparing the input of several sensors (similar to the way events are managed in [56]). Specifically, if the reported system time is 9 am for several consecutive iterations of the MAPE cycle, yet the battery charge level varies significantly, then it can be concluded that one of the sensors is malfunctioning. Both of these represent directions for future research (7.4).

The loosely termed definitions of the **quantitative measurements** for autonomic behaviour evaluation provided in section 2.1.3 have been adapted to fit the SAR problem domain as follows:

- **the quality of the autonomic response** ($u \uparrow$) is measured by counting (over a number of MAPE cycles) the number of executed plans that have driven the SAR system to a state with a higher

---

[9]If monitoring room temperature becomes a requirement, KT will automatically include temperature information in the `State` hierarchy, by reading the values provided by the temperature sensor's API and assigning them to a discrete category, given the provided `thersholds`.

utility than the initial one[10];

- **the cost of autonomy** ($u \downarrow$) is given by the number of discarded plans, over a number of MAPE cycles, that have driven the SAR system to a state with a lower utility than the initial one[11];

- **the speed of the autonomic response** (δt) is defined as the average execution time of the MAPE cycle, over a number of iterations;

- **the degree of proactivity** (p) is the number of decisions the autonomic manager has made without being prompted by a human operator - here, this is evaluated by counting the number of actions considered for inclusion in a plan and averaging the values over a number of MAPE cycles;

- **the learning index** (I_l) will be calculated according to the original definition (equations 2.1, 2.2 and 2.3) with the following interpretation of the symbols:

  - $E(D)$ and $E(PD)$ are equal to each other (all correct learning based decisions are target achievers) and to the number of successfully reused plans (executing a plan from the bank, either directly or after improvement, lead to an increase in utility);

  - $E(D')$ and $E(ND')$ are equal to each other (all incorrect learning based decisions are target damagers) and to the number of unsuccessfully reused plans (executing a plan from the bank, either directly or after improvement, lead to a decrease in utility);

  - $K(PD)$ and $K(ND')$ represent the number of successfully ($p+$) and, respectively, unsuccessfully ($p-$) reused plans from the plan bank;

  - L and M are both 2, as both controlled parameters, brightness and font size, can be learned.

Hence, the learning index of KAS for SAR is computed as follows:

$$I_l = \frac{1}{2} \cdot \frac{p+}{\text{sum}(p+, p-)} \tag{5.1}$$

The data corresponding to the evaluation criteria above is collected under the following conditions:

- A variable number of triggered MAPE cycles[12] is considered for every experiment (that is, every row in Tables 5.5 and 5.6). The event that is used to trigger a new MAPE cycle is a simulated change in the audience focus.

- The experiments are run over two hours, from 10 am to 12 noon, to include 11 am, which is considered late morning from March until October and early morning in the rest of the year.

- The `thresholds` input of the KT algorithm is the one presented in Table 5.1, leading to a separation of the day in four sections: early morning, late morning, early afternoon and late afternoon.

---

[10]This is **not** the number of plans in the plan bank. Some of the successful plans counting towards $u \uparrow$ may have been obtained by improving existing good plans, ultimately replacing them in the bank.

[11]A plan is discarded if the maximum length has been reached and no utility gain was achieved. Apart from this situation, the planning algorithm will continue selecting actions until utility increases.

[12]This is **not** the total number of MAPE cycles. During planning, a MAPE cycle is started after each newly selected action, therefore the *total* number of MAPE cycles is the number of *triggered* MAPE cycles times the number of actions that were considered (including the ones that were eventually discarded).

- The actions (Table 5.3), utilities (Table 5.2) and heuristics (Table 5.4) in the policy document are kept constant throughout the analysis.

- All experiments were run on a Lenovo ThinkPad with an Intel Core i5 CPU @ 2.3 GHz with 8GB of RAM. The KAS for SAR MAPE-K loop is implemented in Java, using the OWL API for interacting with the ontology and Oracle's `ojdbc7.jar`[13] for interfacing with the `Oracle Database 12` manager.

Besides the KAS instance for the SAR problem domain, two control versions of the autonomic system are used for comparison:

- **Trivial.** This is an autonomic system where the manager has exactly the same structure as KAS (Fig. 5.1) apart from the ontology (implying that the reasoner and KT are missing as well). The only knowledge that the manager has access to are the three monitored inputs (since there is no state information stored anywhere, the utilities, candidate plan actions and heuristics are meaningless). The trivial system operates according to two rules, namely "increase the font size and brightness if the focus is less than high" and "decrease brightness if battery charge level is low". The second rule has priority over the first to avoid a conflict when, at the same time, focus is either medium or low and the battery is low. From an intelligent agents point of view [109], the trivial autonomic manager is a *reactive agent* (it merely responds to environmental stimuli), whereas KAS for SAR is a *proactive agent* (it plans a response strategy based on analysis as well as experience and executes it).

- **RDB** (relational database)**.** This is an autonomic system where the manager has exactly the same structure as KAS (Fig. 5.1) with the difference that the ontology is replaced with a relational database storing state information. The normalised database schema is presented in Fig. 5.11.

KAS for SAR compared against the trivial autonomic manager

Experimental setup. The analysis relevant to this scenario is presented in Table 5.5, where each row represents a separate experiment. An experiment consists in running the trivial autonomic manager as well as the KAS instance in the SAR problem domain, over a number of explicitly triggered MAPE cycles within the considered two hour time range. The MAPE cycle triggers are simulated changes in the audience focus (for instance, the **T** value on the first row of the table means that the level of audience focus was changed 4 times between 10 am and 12 noon, with pseudorandom periodicity and to pseudorandomly selected levels). The battery started at full charge and did not drop under 75% in the two hours, hence the only other MAPE loop trigger, besides the focus variations, is at 11 am, when the time of day changes from early to late morning (experiments were run in March). The experiments are meant to compare the reactive (trivial system) and proactive (KAS instance) approaches to change, in terms of the number of successful ($u \uparrow$) and unsuccessful ($u \downarrow$) plans executed and, respectively, discarded. The average duration of a MAPE cycle over the length of each experiment (δt) is provided just for the KAS instance, as the trivial version entails a minimum computation load (the analysis process consists, exclusively, in deciding which of the two rules to apply), therefore terminates very fast.

---

[13]http://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.html

HourInterval
- Id
- Start
- Finish
- **primary key**: Id

Link
- HourId
- MonthId
- TimeOfDayId
- **primary key**: (HourId, MonthId)
- **foreign key**: HourId **to** Hour(Id)
- **foreign key**: MonthId **to** Month(Id)
- **foreign key**: TimeOfDayId **to** TimeOfDay(Id)

TimeOfDay
- Id
- Name
- **primary key**: Id

MonthInterval
- Id
- Start
- Finish
- **primary key**: Id

State
- Id
- Focus
- Battery
- TimeOfDayId
- Utility
- **primary key**: Id
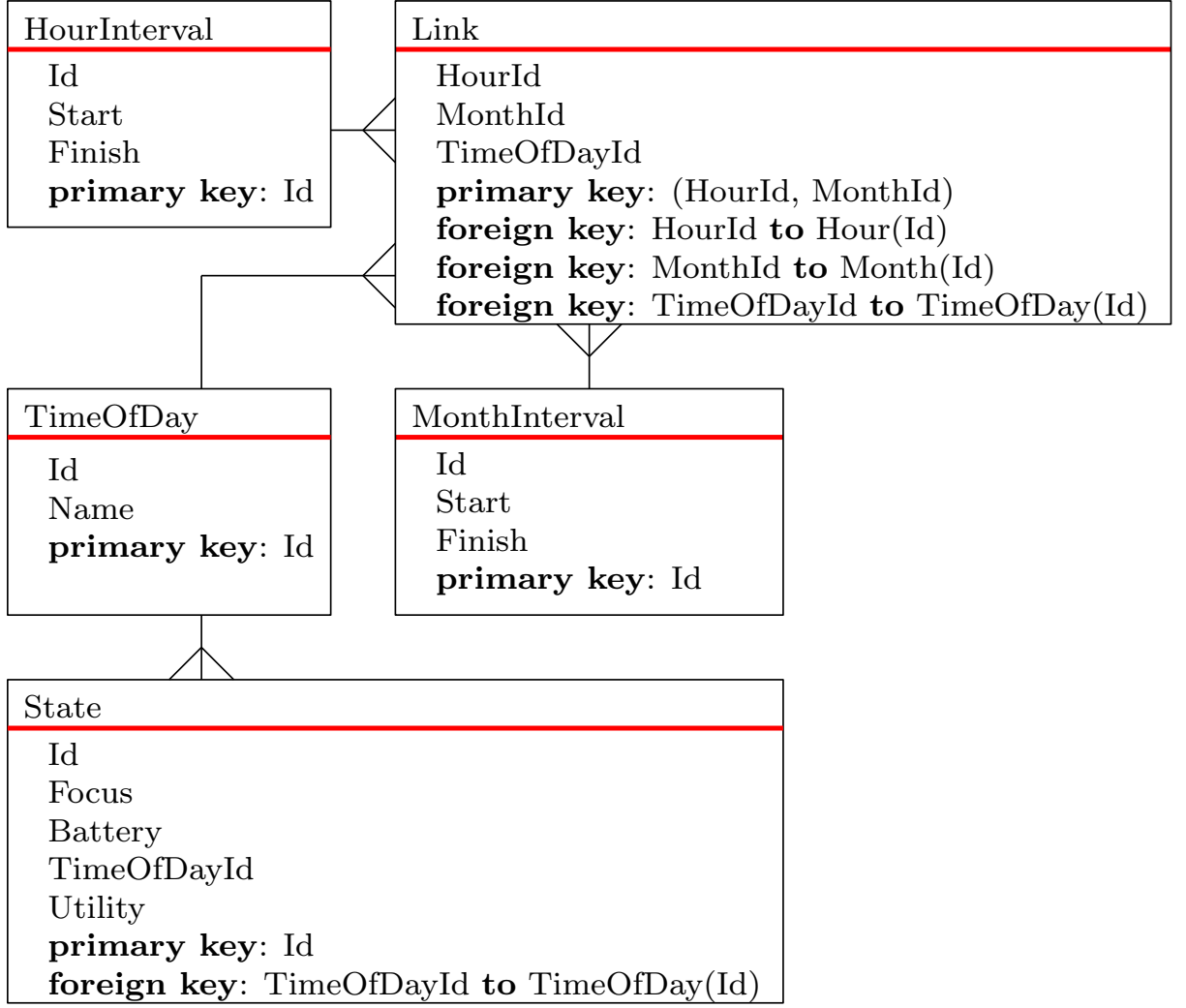- **foreign key**: TimeOfDayId **to** TimeOfDay(Id)

Fig. 5.11: A partial Entity Relationship Diagram describing the database equivalent to the Onto++ instance for SAR

Results' discussion. The first three experiments confirm the intuitive expectation: the trivial autonomic manager occasionally increases utility by applying one of the two "plans" hard-coded in the rules. This happens by chance, as there is no correlation between the reactive agent's success rate and the number of triggers. On the other hand, the proactive agent only generates utility increasing plans ($u \downarrow$ is 0 for the first three experiments). This is a consequence of the logic behind the `plan` algorithm that does not stop before generating a set of actions that brings the managed resource to a state with a higher utility than the initial one (line 11 in Table 4.5). This changes with the fourth experiment, where the KAS instance discards 12 plans out of the 2000 generated in response to the triggers. It follows that in those 12 cases, the plans reached the imposed maximum length (`L = 10` in Table 4.5) without achieving an increase in utility. The intuition behind this result is that the frequency of focus changes is too high (once every 3.6s), thus preventing new actions included in the plan from building up on the effect of previous ones. This is confirmed by the final experiment, where doubling the focus change frequency causes the proactive agent to dismiss most of its plans. Taking into account that the duration of KAS's MAPE loop (δt) is around 360ms and that the maximum plan length is 10, the system has a maximum of 3.6s to build a successful plan. In experiment four, that coincides with the duration between

two simulated focus changes. It follows that injecting a change in the system in the final stages of plan construction may compromise the effect of previously inserted actions, causing the entire plan to fail. Further increasing the frequency of focus changes (experiment five) causes the proactive agent to stop operating effectively. However, the reactive agent performs significantly better in this last scenario (twice as many successful plans than what the KAS instance generated). This needs more experimentation, but does bring about the posibility that a reactive approach may be more effective than a proactive one, given a highly dynamic environment. Of course, experiments four and five are not realistic in the SAR scenario (it is more likely to suspect a camera malfunction than accept that the audience's focus changes every 3.6s). Regardless, these experiments were included given the intriguing insight they provide with respect to what is effective change management in highly dynamic environments.

Table 5.5: KAS for SAR and trivial autonomic manager comparison
plan length capped at 10 actions

	trivial		**KAS for SAR**		
T	$u\uparrow$	$u\downarrow$	$u\uparrow$	$u\downarrow$	δt[ms]
5	1	4	5	0	340
20	0	20	20	0	390
100	22	78	100	0	350
2000	103	1897	1988	12	355
4000	301	3969	150	3850	360

KAS for SAR compared against the database-supported autonomic manager

Experimental setup. The data relevant to this comparative study is presented in Table 5.6. The control version that KAS for SAR is compared against is an autonomic manager employing a relational database (Fig. 5.11) for knowledge management. The comparison between the database and the Onto++ instance, within an autonomic context, is justified by the following reasons.

- Databases are, at present, the industry standard solution for managing knowledge and there has been significant debate whether ontologies can match up to the status quo [126, 160].

- Databases operate under the closed world assumption (CWA), namely every fact that is not explicitly stated in the database tables is *false*. Ontologies employ an open world assumption (OWA), where the truth value of a fact that is not explicitly stated as a semantic property is *unknown*. This allows the reasoner to infer implicit knowledge and proactively insert it in the ontology, thus facilitating the learning process.

- A change in the requirements (for instance, adding two times of day, midmorning and midafternoon, to the existing four) would require coordinated updates to several database tables (to satisfy foreign key constraints). The ontology would only require the addition of two concepts, that the reasoner would automatically connect to the existing classes via semantic subsumption (specific examples are provided in the summary section).

Besides $u\uparrow$, $u\downarrow$ and δt, also considered in the previous comparison, Table 5.6 provides experimentally collected values for p, the average number of actions considered during planning (both the ones included

in plans and the ones discarded), $p+$ and $p-$, the number of successfully reused and, respectively, discarded plans and I_l, the learning index interpreted for SAR in equation 5.1. As far as the number of triggers is concerned, this reflects focus changes (simulated via the pseudorandom generator in java, as in the previous comparison), one change from early morning to late morning and two battery changes (started at 80%, dropped to 70% at some point and then got charged back up to 100%).

Results' discussion. The first three experiments where no upper limit was imposed to plan length (Table 5.6a) reveal that both autonomic systems build successful plans in response to all changes in the managed resource. The fourth experiment shows no discarded plans ($u \downarrow = 0$), however, the number of successful plans ($u \uparrow$) does not match up to the number of triggers. Indeed, 9 and 11 triggers did not receive any response from KAS for SAR and the database supported system, respectively. It is known, from the previous comparison, that 2000 triggers over two hours give the MAPE loop enough time to find good plans with a maximum length of 10 actions. Since there is no upper limit to the plan length, the planner will continue adding actions to the current plan in an attempt to increase utility (p is 157 and 180, respectively, very large values compared to previous experiments), an effort continually "sabotaged" by the intense dynamics of the managed resource. This shows the importance of setting an upper plan size limit in order to improve the planner's capacity to adapt to frequent change. To confirm, the second and third rounds of experiments (with the maximum plan length set to 5 and 10, respectively) show that all triggers are accounted for, with $u \uparrow$ and $u \downarrow$ adding up to the expected amount. Note that the optimal value for the maximum plan length (L in Table 4.5) can be configured by running experiments in the "threshold" trigger zone (namely, 2000). Nonetheless, this is a limitation of the KAS architecture, as the plan size limit parameter, although experimentally configurable, is problem dependent.

As shown by the average duration of the MAPE cycle (δt), the database supported system takes longer than KAS for SAR. This is due to the fact that retrieving the current state (an analysis step) requires a complicated SQL query with joins over all database tables (see summary section). This turns out to be more computationally costly (although not by a significant amount) than the reasoner performed tasks of creating the `CurrentTime` (Fig. 5.7), `CurrentBattery` and `CurrentFocus` (Fig. 5.8) concepts from sensor data, subsuming them under the correct `TimeOfDay` (Fig. 5.4), `Battery` and `Focus` subclasses and matching them against `State` fillers (Fig. 5.3) to determine the current utility.

The number of successfully reused plans ($p+$) from the plan bank increases, alongside the number of discarded plans ($p-$), with the frequency of triggers. This strong correlation shows that the learning capability exhibited by the autonomic manager, irrespective of the underlying knowledge platform, is consistent. Plan reuse factors into the learning index as defined in the SAR context (equation 5.1). This formulation implies that the maximum value for I_l is 0.5, which is achieved in all realistic scenarios (apart from experiments involving a poorly calibrated upper limit for plan length).

Quantitative evaluation summary.

The comparison of KAS for SAR against the trivial autonomic manager (a reactive agent with no knowledge of its own state) reveals the existence of a "threshold" with respect to the system's tolerance to the level of variation in the monitored inputs. If the number of MAPE loop triggers exceeds this threshold (given by the average duration of the autonomic control cycle times the maximum plan length), the manager can no longer respond to change effectively. In addition, as the second series of experiments shows, the plan length limit can be experimentally calibrated by testing the autonomic system in the "threshold" range.

Table 5.6: KAS for SAR and database-supported autonomic manager comparison

(a) no maximum plan length

| | KAS for SAR | | | | | | | RDB | | | | | | |
T	$u\uparrow$	$u\downarrow$	δt	p	$p+$	$p-$	I_l	$u\uparrow$	$u\downarrow$	δt	p	$p+$	$p-$	I_l
5	5	0	375	7.3	2	0	0.5	5	0	350	8	1	0	0.5
20	20	0	350	15.3	5	0	0.5	20	0	400	16.1	3	0	0.5
100	100	0	365	12.1	11	0	0.5	100	0	410	11.9	9	0	0.5
2000	1991	0	355	157	75	0	0.46	1989	0	405	180	77	0	0.34

(b) maximum plan length is 5

| | KAS for SAR | | | | | | | RDB | | | | | | |
T	$u\uparrow$	$u\downarrow$	δt	p	$p+$	$p-$	I_l	$u\uparrow$	$u\downarrow$	δt	p	$p+$	$p-$	I_l
5	5	0	352	6.8	3	0	0.5	5	0	376	9.2	0	0	0
20	18	2	360	17.2	3	2	0.3	17	3	432	18.3	0	2	0
100	90	10	355	15.2	9	3	0.37	88	12	402	10.3	8	5	0.31
2000	1507	493	323	12.3	9	25	0.13	1486	514	399	12.7	2	12	0.07

(c) maximum plan length is 10

| | KAS for SAR | | | | | | | RDB | | | | | | |
T	$u\uparrow$	$u\downarrow$	δt	p	$p+$	$p-$	I_l	$u\uparrow$	$u\downarrow$	δt	p	$p+$	$p-$	I_l
5	5	0	331	7.1	2	0	0.5	5	0	392	11.1	2	0	0.5
20	20	0	343	12.3	3	0	0.5	20	0	390	15.3	1	0	0.5
100	100	0	390	13.2	8	1	0.44	100	0	435	12.1	5	2	0.35
2000	1991	9	315	13	13	7	0.32	1992	8	411	13.2	14	8	0.32

The comparison of KAS for SAR against the database-supported autonomic manager yields several conclusions with respect to the superiority of ontologies over relational databases.

- Databases do not support unsupervised learning. During knowledge extraction, facts that the reasoner can automatically infer, such as "a battery charge level of 73% is medium", have to be explicitly asserted by adding the appropriate entries to three database tables (`BatteryLevel`, `Battery` and `BatteryLink`), while making sure that the foreign key restrictions (Fig. 5.11) are complied with.

- A minor change in the state model, e.g., the addition of midmorning and midafternoon, would not only imply adding records to three separate tables (`HourInterval`, `TimeOfDay` and `State`) but also updating `TimeOfDayLink` to ensure compliance with Codd's three normal forms [51]. This rigidity of the relational database format is contrasted by the flexibility of the ontology, where the rightful place (under ontology class `TimeOfDay`) of the two new concepts in the `Entity` hierarchy will be inferred by the reasoner.

- Another consequence of the inflexible structure of relational databases is the syntactic complexity of the SQL query required to retrieve the current system state at the start of every MAPE loop.

Besides the type of learning enabled by the reasoner (namely, unsupervised inference of implicit facts), the autonomic control loop superimposes another layer: learning from planning experience.

The autonomic manager's effectiveness at recycling plans is illustrated by the fact that the number of target achievers ($p+$) is consistently greater than that of target damagers ($p-$) (except for "threshold" experiments), leading to the maximum achievable (in the SAR application domain) learning index value ($I_l = 0.5$). The fact that the theoretical upper limit of the learning index ($I_l = 1$) is not reached can be traced back to the structural simplicity of KAS for SAR rather than a limitation in its learning capacity. Indeed, in order for a system to have the potential of achieving a learning index of 1, the number of successful learning decisions should exceed that of target achievers, whereas unsuccessful learning decisions should only count towards a fraction of the target damagers (see equation 2.1).

5.4.2 Semantic Components Evaluation

The chief benefits that autonomic managers, especially the analysis module, draw from the use of semantic tools for knowledge management have been presented in section 5.4.1. The flexibility of ontologies over the rigidity of relational databases as well as the way unsupervised reasoning (inference of implicit facts) supports learning within the MAPE loop have been discussed and evaluated in the broader context of testing KAS's autonomic components. However, some qualitative and quantitative metrics available in the ontology related literature (and covered in 2.2.5) may reveal other facets of the semantic-autonomic interaction.

Qualitative Evaluation

ONTOMETRIC. This platform measures ontology flexibility and expressivity by evaluating structural elements such as the number of concept instances and the presence of n-ary relationships. At first sight, the ontology supporting KAS for SAR would not fare well with respect to ONTOMETRIC criteria, however, not without reason.

- There are no explicitly asserted individuals in the SAR ontology, however, that does not imply that all concepts are abstract. On the contrary, the ontology does store specific domain entities, only they are modelled as classes rather than instances, to allow for more powerful reasoning and added flexibility. To illustrate, let us consider the `State` hierarchy (Fig. 5.3). In terms of concept granularity, all `State` subconcepts should be individuals as they represent specific system states that cannot be subsumed by other classes. However, bear in mind that the ontology is dynamically generated by the KT algoritm (Table 4.1) that is designed to cater to a wide variety of application domains. Although the two tier state model (with one root concept and 36 leaf classes) is sufficiently accurate for SAR, other managed resources may require a state hierarchy with several levels (e.g., in a smart home, the `OnlineState` passed on to power suppliers over the internet is different than the `OfflineState` accessed for controlling room brightness by closing/opening the shutters - an intermediate `State` hierarchy level is needed to hold the two classes). To maintain the cross-field applicability of KAS as well as avoid unnecessarily complicated logic in the underlying algorithms (KT in particular), Onto++ instances do not contain individuals[14].

---

[14]This is not to say that using the presence of individuals to evaluate ontology flexibility is ill-advised, just that this criterion applies better to ontologies as stand-alone artefacts (designed to serve as thesauri of knowledge) than to ontologies as components supporting an autonomic infrastructure.

- The Onto++ instance for the SAR problem domain does not feature any n-ary relationships (all properties can be modelled with standard RDF triples). However, KT is capable of reification (4.3.2), as illustrated in the second application scenario (chapter 6). In a broader interpretation, the lack of n-ary relationships in the SAR model (where they are not needed) is proof of KT's flexibility, as it is capable to generate simple or more complex ontologies, given the intricacy of the problem domain.

OOPS! A simple yet highly informative online tool, OOPS! provides a helpful ontology diagnosis based on several good practices distilled from known ontology design patterns. Specifically, the tool is capable of detecting common structural "pitfalls"[15] (maintained in a catalogue that users can help extend) such as the presence of cycles, orphans (disconnected concepts or unused properties) or equivalent classes in lieu of synonyms. Initially, OOPS! revealed three issues with the Onto++ instance modelling the SAR domain, as can be seen in Fig. 5.12a [16].

- **Missing disjointness.** In OWL ontologies, it is generally good practice to declare as disjoint those subclasses that exhaustively cover the domain modelled by their superclass without overlapping [136, 82]. This should be the case for all leaf level `State` classes as well as `TimeOfDay`, `Battery` and `Focus` subconcepts, respectively. Although correctly generated by KT, the final three hierarchies were missing the appropriate disjointness axioms.

- **Inverse relationships not explicitly declared.** This problem was located in the `State` hierarchy: every leaf level state concept is connected via the appropriate properties to four fillers (one for each monitored input and one for the state utility). Since properties `hasTimeOfDay`, `hasFocus` and `hasBattery` are not transitive, it is good practice to assert an inverse property for each. For instance, the inverse property of `hasFocus` would connect `lowFocus` to `State` subconcepts 1 through 12 (see Table 5.2). This is a supplementary verification mechanism to ensure that the fillers of the direct properties are correctly set.

- **No licence declared.** The Creative Commons licence declaration[17] was missing from the ontology metadata.

After making the necessary modifications to the KT algorithm in order to fix the problems above, the Onto++ instance for SAR meets all OOPS! evaluation criteria (Fig. 5.12b).

Quantitative Evaluation

Basic metrics represent counts of ontology elements (such as the number of concepts, equivalence axioms, inverse properties, etc.). This information is easily obtainable in editors such as Protege (the `Ontology metrics` tab) and are not particulary relevant in terms of assessing the structural soundness of the ontology (different problem domains require different numbers of concepts and prperties to model them). What is more important than the number of ontology elements is their very *existence* (e.g., disjointness restrictions should be in place for all relevant concept hierarchies, regardless of how many

---

[15]For a full list, see `http://oops.linkeddata.es/catalogue.jsp`.

[16]Retrieved from `http://oops.linkeddata.es/`.

[17]https://creativecommons.org/licenses/by/4.0/

(a) Initial diagnosis

(b) Final diagnosis

Fig. 5.12: OOPS! diagnoses for the SAR ontology - before and after corrections

there actually are) and that is checked by online tools such as OOPS!. Moreover, these counts become semantically relevant [181] only when the ontology is normalised, which the Onto++ instance for SAR is not (the one normalisation condition - see 2.1.3 for a full list - that is not met is the instantiation of all leaf level concepts for reasons explained earlier in the qualitative evaluation section). Some structural ontology metrics that do show promise are the ones used to predict reasoning efficiency [96]. However, the reasoner's impact on the autonomic loop's execution time has already been measured (δt) in the quantitative analysis section of 5.4.1[18].

Temporal and category bias are sound indicators of an ontology's accuracy in representing a changing domain. Category bias, as defined in [79], does not apply to the Onto++ instance for SAR, as there are no multiple levels of representation. Temporal bias is relevant only to a certain extent: the state model can indeed be represented as a vector over three dimensions (one for each monitored input). However, the continuous values of the focus, battery and time signals are discretised in the ontology, making it mathematically impossible to calculate the cosine similarity between the model and the modelled vectors. Since the ontology is dynamically generated by KT, a change in the modelled domain (such as the addition of midmorning and midafternoon considered earlier) would be seamlessly integrated in the `State` concept hierarchy (provided that the `thresholds` parameter is appropriately ammended). This supports the claim that the Onto++ instance provides an accurate representation of the (changing) SAR problem domain.

Evaluating the cost of the ontology engineering process requires application developers and/or domain experts to set weights representing the importance (cost) of each OE stage with respect to the development of the application that the ontology supports. Since Onto++ instances are automatically generated by KT, the cost of the OE process is more relevantly evaluated by KT's computational

---

[18]This explanation as to why basic metrics were **not** included in this section is given since many reasearchers [133, 135, 68] still use them to evaluate their ontologies.

complexity (equation 4.1).

Summary

The highlight of this discussion around ontology evaluation is the fact that the Onto++ instance for SAR meets a wide variety of common good design criteria (Fig. 5.12b). This is an argument that supports the efficiency of KT. The design of the Onto++ structural template (that the SAR instance is created from) has been evaluated from a different standpoint (ATAM) in section 4.5. Another relevant aspect is the compliance of the Onto++ instance development process with a widely accepted OE methodology, namely NeON (this is detailed in the architecture description section 5.2.1).

Autonomic Systems with State-less Ontologies

The second practical implementation of KAS is deployed in the career support problem domain. The managed resource is the online career knowledge space, scattered across various web resources (company and education providers' websites, centralised job openings repositories, general information about professions published on webpages with other, non-related content, etc.) and currently processed, to a very limited extent, by traditional job search engines. The KAS instance used in this context does not maintain a state hierarchy, as the volume of available data would make that computationally infeasible. Instead, the Onto++ instance initially models domain expert knowledge and is updated by system users in subsequent iterations. The description of the KAS instance in the careers domain will follow the same steps as in the case of the SAR problem space.

6.1 Application Domain

The **C**areer **D**ecision **S**upport (CDS) problem implies providing interested users with a *complete*[1], *correct* and *intuitive* compilation of knowledge, relevant to their career related query.

For a better understanding, consider the following motivating scenario involving a user interested in a software engineering career. Finding online information about this profession would imply:

- searching the web for a general definition (National Careers Service[2] and Wikipedia are popular resources)

- browsing online education resources (HESA[3]) to find the required academic qualifications necessary to be eligible for a job in the field

- getting a list of the relevant job openings (for instance, from `indeed.co.uk`)

- manually centralising the data acquired in the previous three steps and matching it against the user's personal profile (academic degree and professional interests).

When using the KAS instance for the CDS problem domain, the experience is significantly different. By simply typing "software engineering" in a search box, the same user would, at the click of a search button, get access to:

---

[1]Within the limits of the available knowledge sources.

[2]`https://nationalcareersservice.direct.gov.uk/search/pages/JobProfileResults.aspx?k=software%20engineer`

[3]`https://www.hesa.ac.uk/jacs/`

- a visual graph showing the requested career as the central node as well as related ones (e.g., information engineering, software development, etc.) - this provides some much needed perspective by indicating the place that software engineering occupies in the broader field of careers

- a pie menu (visible by right clicking any graph node) with links to relevant jobs (`indeed.co.uk` and general Wikipedia information)

- additional, useful data, such as a list of web resources relevant to software engineering that other users have visited and rated (the scores are displayed as well).

Thus, the KAS instance addresses the core requirements of the CDS problem as follows (these are relevant to the second research question in 1.2, specifically objectives O1.3 and O2.1):

- **Completeness.** Information from all sources monitored by the system (job search engines, other users' reviews, etc.) is automatically compiled, clustered on the relevant graph node and accessible via a pie menu.

- **Correctness.** The way careers nodes relate to each other is regulated by a semantic reasoner (that detects logical inconsistencies such as software development being both the child and the parent of software engineering).

- **Intuitiveness.** The relevant career information is presented in a graph where the connections between careers can be visually explored. This manner of displaying results is more insightful and natural to humans than the list format produced by traditional job search engines.

To sum up, the *managed resource* (Fig. 6.1) in the CDS scenario comprises a portion of the online space of career resources (including the users exploring it) that is monitored through the following channels:

M1 job postings from employment search engines

M2 tags (semantic annotations) and reviews used/ written by registered users

M3 edits performed on the careers graph by registered users

M4 search keywords (queries) provided by registered users.

The view (screen or results display window, also part of the managed resource) containing the collection of knowledge relevant to a user query is modified in terms of the

E1 displayed content: a correct, complete and intuitive *graph* that replaces the *list* of available jobs returned by traditional search engines and the other heterogeneous information that the user would have to retrieve and analyse manually.

Another important advantage of exploring career graphs rather than job lists is that subtle connections become significantly easier to observe. For instance, starting from the graph produced in the motivating scenario , the user can quickly discover that digital arts is the twice removed "cousin" of software engineering (via software development and software systems for the arts). Consider the amount of time and effort it would have taken to discover this connection via traditional career research.
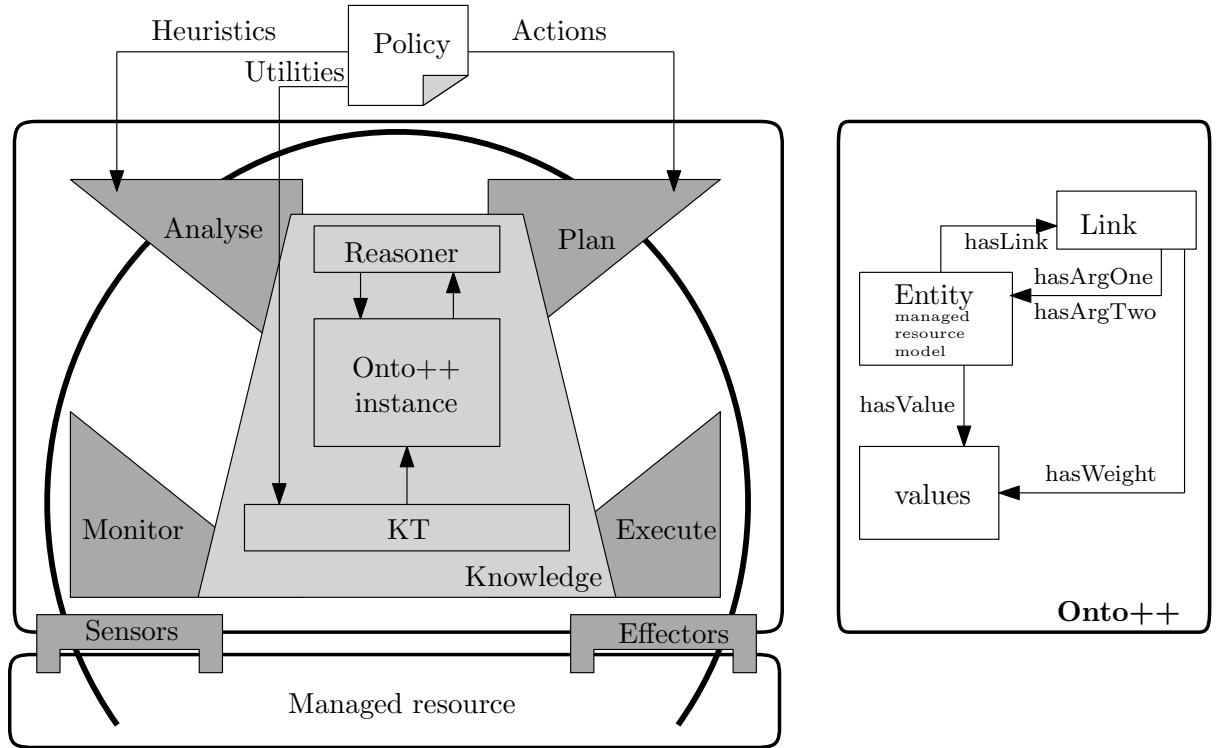
6.2 Architecture Description

Fig. 6.1: KAS with state-less ontology - component view

6.2.1 CDS Ontology and KT

As in the case of SAR, the CDS ontology learning process is best mapped against the NeOn methodology [168], namely scenarios 2 (OL via reusing and re-engineering non-ontological resources) and 8 (OL by means of restructuring/ extending ontological resources).

Non-ontological resource selection

Various career related knowledge sources were considered, assessed (in terms of coverage, precision and consensus) and selected[4] to provide input for the CDS ontology. Those were:

- the domain expert's knowledge about relationships (inheritance, equivalence and synonymy) between careers

- data from education providers and statistics agencies (JACS and SOC codes[5])

- online job listing repositories (`indeed.co.uk`).

---

[4]These tasks were performed by the careers expert, Lord Ralph Lucas, Lucas Publishing, Good Careers Guide.

[5]`http://www.neighbourhood.statistics.gov.uk/HTMLDocs/dev3/ONS_SOC_hierarchy_view.html`

Non-ontological resource reverse engineering and transformation

Knowledge from all previously selected sources was compiled in a spreadsheet[6] and reverse engineered to detect patterns in the data. Each spreadsheet row was found to contain the name of one career concept along with all related career concepts, the relationship type (parent, child, sibling or synonym) and the relationship strength (a number between 0 and 100). The example at the top of Fig. 6.2 illustrates this pattern: it states that biomolecular sciences is the child of sciences with a strength of 100, the sibling of biology with a strength of 50 and the synonym of bio-molecular sciences (an alternative spelling used for disambiguation). This analysis of the data in the input document revealed two aspects: the available knowledge is discrete (there are no continuous signals in the managed resource as in the case of SAR) and properties are three-faceted (every sibling and inheritance relationship between two careers is weighted by a numerical value called strength).

Entity	Synonym of	Link to	Link type	Link strength	Link to 2	Link type 2	Link strength 2
BIOMOLECULAR SCIENCES		SCIENCES	CHILD	100	BIOLOGY	SIBLING	50
BIO-MOLECULAR SCIENCES	BIOMOLECULAR SCIENCES						
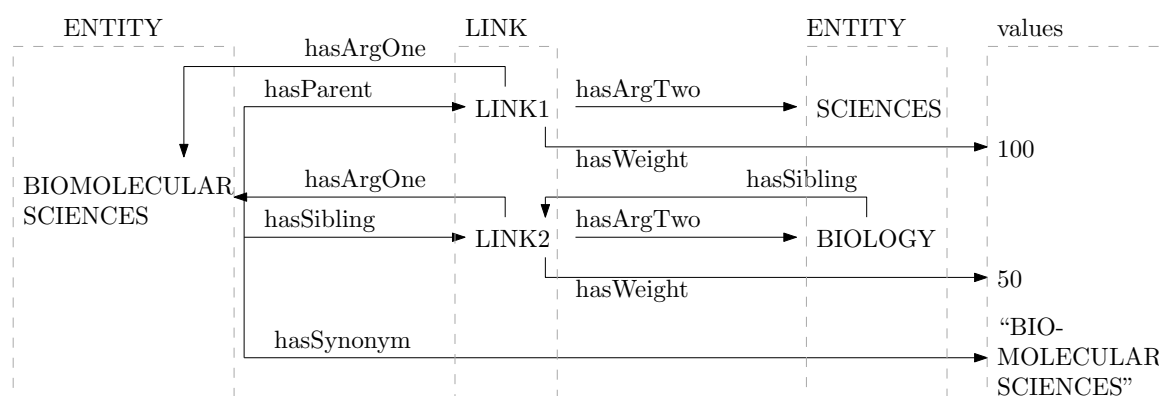
Fig. 6.2: Reification in the CDS ontology

The transformation of the spreadsheet knowledge into a format that could be represented in an ontology yielded a new conceptual model (the Onto++ structure in Fig. 6.1). The `State` hierarchy was excluded since it is not applicable to the CDS domain and the `hasProperty` relationship (that used to connect `Entity` subconcepts to each other in the SAR domain) has been reified with the help of the `Link` hierarchy (Fig. 6.2). A full description of the conceptual model supporting the CDS ontology is provided below.

- The `Entity` hierachy stores concepts describing careers (such as `BIOMOLECULAR SCIENCES` in Fig. 6.2). The definition (in Machester OWL syntax) of the entity hierarchy root is presented in Fig. 6.3.

- The `hasValue` property connects an `Entity` concept to a constant value (a number or a string). It has three subproperties, `hasSynonym`, `hasJACS` and `hasSOC`. The first two have a filler of type string, namely the domain entity's synonym (an alernate spelling, such as `"BIO-MOLECULAR SCIENCES"` in the lower right hand side corner of Fig. 6.2) and, respectively,

---

[6]Autored by Lord Ralph Lucas, Lucas Publishing, Good Careers Guide.

```
Class: <CDSonto#Entity> EquivalentTo:
  <CDSonto#hasParent> some <CDSonto#Link>
  and (<CDSonto#hasSibling> some <CDSonto#Link>)
  and (<CDSonto#hasSynonym> some xsd:string)
  and (<CDSonto#hasJACS> some xsd:string)
  and (<CDSonto#hasSOC> some xsd:integer)
```

Fig. 6.3: `Entity` concept definition

the associated JACS value (an alphanumeric identifier that is C760[7] for biomolecular science). Property `hasSOC` has a filler of type integer, repesenting the numeric SOC code.

- The `Link` hierarchy contains abstract concepts (in the sense that they do not directly model entities from the problem domain) used to reify properties with strengths (Fig. 6.2). The definition of the link hierarchy root is presented in Fig. 6.4.

```
Class: <CDSonto#Link> EquivalentTo:
  <CDSonto#hasArgOne> some <CDSonto#Entity>
  and (<CDSonto#hasArgTwo> some <CDSonto#Entity>)
  and (<CDSonto#hasWeight> some xsd:integer)
```

Fig. 6.4: `Link` concept definition

- The `hasLink` property connects `Entity` hierarchy concepts to `Link` classes. There are two subproperties, `hasParent` and `hasSibling`, both multifaceted (with strengths) and in need of reification (illustrated in Fig. 6.2 on the `BIOMOLECULAR SCIENCES` example).

- The `hasArgOne` and `hasArgTwo` properties connect link concepts to fillers from the `Entity` hierarchy. `hasArgOne` and `hasArgTwo` are the inverse properties of `hasLink`.

- The `hasWeight` property connects link concepts to numerical fillers, namely the strengths associated to reified properties. All constants (numbers and strings) are represented by block `values` in Fig. 6.1.

A note about the downside of reification: the `hasParent` property is unavoidably duplicated - implied in the OWL inheritance hierarchy and explicitly asserted as a reified property. Moreover, the additional `Link` hierarchy and its associated properties significantly inflate the ontology. Consequently, there is a compromise to be made between ontology size (will most likely impact reasoning, therefore learning, speed) and domain representation accuracy.

Ontology forward engineering

The previously formulated conceptual model is automatically generated by KT (Knowledge Translator), the ontology learning algorithm described in 4.3.2. The algorithm's inputs and outputs are configured as follows.

---

[7]Source: `https://hesa.ac.uk/component/content/article?id=102&ItemId=136&limit=1&start=6`

Input `inputFile`. This represents the spreadsheet containing the initial knowledge compiled from various sources by the domain expert. The `Entity` and `Link` hierarchies will be created according to the logic on lines 14-22 in Table 4.1.

Input `isDiscrete`. The KAS instance for CDS monitors discrete events (**M1**, **M2**, **M3** and **M4** introduced in 6.1). The `inputFile` that the initial version of the Onto++ instance is created from also contains discrete concepts and properties. Thus, the value of this input is `true`.

Input `thresholds`. Since no discretisation is performed, this parameter is `null`.

Output `o`. KT will return an ontology, modelling the CDS problem space, in full compliance with the Onto++ template (that is, comprising the `Entity` and `Link` concept hierarchies, along with their associated properties, as shown in Fig. 6.1).

Ontology restructuring and extension

This is performed by registered system users, several times throughout the ontology life cycle. Once generated (by KT) the careers' ontology is exposed to the public and can be edited through a graphical interface (a feature that is not available, nor necessary, in SAR). This allows the autonomic manager's knowledge to reflect the community consensus with respect to the careers' domain and not just the domain expert's view. However, to comply with the correctness component of the system goal (6.1), user edits are only considered pending reasoner verification.

6.2.2 Reasoner

The reasoning functions performed on the Onto++ instance for CDS are explained in the following.

Subsumption. During the ontology life cycle, specifically the forward engineering (KT) and extension (editing) stages, the reasoner infers the correct hierarchy (`Entity` or `Link`) that new concepts should be included in. For instance, concept BIOMOLECULAR SCIENCE (Fig. 6.5) matches the definition of `Entity` (Fig. 6.3[8]), therefore will be subsumed as one of its subconcepts.

```
Class: <CDSonto#BIOMOLECULAR SCIENCE> EquivalentTo:
  <CDSonto#SCIENCE>
  and (<CDSonto#hasParent> some <CDSonto#Link1>)
  and (<CDSonto#hasSibling> some <CDSonto#Link2>)
  and (<CDSonto#hasSynonym>
    some xsd:string[>= "BIO-MOLECULAR SCIENCES", <= "BIO-MOLECULAR SCIENCES"])
  and (<CDSonto#hasJACS> some xsd:string[>= "C760", <= "C760"])
  and (<CDSonto#hasSOC> some xsd:integer[>= 2112, <= 2112])
```

Fig. 6.5: BIOMOLECULAR SCIENCE concept definition

Apart from the immediate benefit of maintaining a well structured ontology, subsumption also supports learning, albeit in a different way than in the case of SAR (see 6.2.4 for details).

Querying. In the CDS context, the reasoner performs a query in the following situations:

---

[8]The fact that exact fillers have to be sprcified as ranges is a downside of using classes for what should, intuitively, be modelled as individuals. This is a necessary compromise to allow subsumption.

- when the user initiates a new search (**M4**): the search keyword will be used to formulate a SPARQL query (Fig. 6.6 shows the one for `BIOMOLECULAR SCIENCE`) and run it to find all `Link` concepts the targeted entity is related to (be it via `hasParent` or `hasSibling`). More Java code is in place to unpack the retrieved `Link` objects and extract the actual parents and siblings. The ones connected to the targeted entity via no more than two sibling or parent links are displayed in the visualiser (**E1**) (see 6.3 for a screen shot of the result).

```
PREFIX cds: <http://xmlns.com/CDS>
SELECT ?link1 ?link2
WHERE {
  cds:BIOMOLECULAR SCIENCE cds:hasParent ?link1 .
  cds:BIOMOLECULAR SCIENCE cds:hasSibling ?link2 .
}
```

Fig. 6.6: A SPARQL query to support searching the CDS ontology

- when the user tags online resources (**M2**) with concepts from the ontology (using a browser add-on described in 6.3): the full set of tags used by a registered explorer are fed into a set of SPARQL queries (similar to the one in Fig. 6.6). The graphs produced by running the batch query (and performing the previously mentioned Java post-processing) are displayed in the visualiser (**E1**) to form the user's *personal ontology* (a visual expression of the CDS ontology segment the user has shown an interst in while browsing).

Consistency verification. When users edit the CDS ontology (by adding, deleting or modifying the definitions of concepts and properties), their changes are stored in a temporary copy of the ontology and published only if the reasoner's consistency check is successful.

6.2.3 Policy

The CDS problem domain does not have a state model, therefore utilities are not necessary. In its current implementation, the CDS autonomic manager is not guided by heuristics. Hence, the only policy document section that applied to the careers' space is the one storing plan actions (Table 6.1). They are:

- segment: run the SPARQL query modelling the search initiated by the user and retrieve the relevant ontology segment

- edit: process the deletion/ addition/ other modification that the user has operated through the interface and store it in a temporary copy of the ontology

- tailor: run the set of SPARQL queries built around the tags used as annotations by the current explorer and retrieve the relevant ontology segments

- display: populate the visualiser with the result of the most recent action (either a simple query or a batch one)

- classify: check the logical validity of the ontology and, if confirmed, publish it (commit all pending changes).

Table 6.1: CDS policy document - plan actions

a1	segment
a2	edit
a3	tailor
a4	display
a5	classify

Note that all actions apart from **a4** involve the reasoner, which is proof the the strong impact this semantic tool has on the planning stage of the MAPE loop.

6.2.4 Monitor, Execute and the Managed Resource

The system's monitor component comprises several APIs that pass sensor data to the analysis module. There is a software sensor for each of the monitored spaces where career relevant information is posted or used: a web service detects job postings from `indeed.co.uk` relevant to a given ontology concept (**M1**), a browser add-on collects tags from user career webpage annotations (**M2**), a web interface records user edits (**M3**) and search keywords (**M4**). The execute module is a JavaScript API that communicates the ontology segment(s) to be displayed to the effector, a Cytoscape [163] plugin (**E1**)[9].

Learning in the CDS context is done *explicitly*, via **M2** (without the user being aware) and **M3** (directly by the user through the web interface). The process is similar to Amazon's mechanical turk principle[10], only the participating humans' reward is not money but the career expertise they gain by exploring and tagging web resources and inherently feeding that knowledge into the CDS ontology. The learning process is completed when the reasoner verifies and subsumes user provided knowledge in the `Entity` hierarchy (as explained in 6.2.2). It is intersting to observe that this is different from the *implicit* way learning is performed in SAR (5.2.2). It can thus be concluded that the KAS architecture is capable of supporting different learning styles.

6.2.5 Analyse

The `analyse` algorithm is presented in Table 4.3. The inputs and outputs that apply to the CDS context, as well as the way they are processed throughout analysis, are explained in the following.

Input `o`. This parameter is `null`, as no changes are made to the ontology during monitoring. In the CDS context, method `findCrtState` (line 1 in Table 4.3) does not refer to the state of the managed resource in the SAR sense, but to the active changes (detected and still unanswered) in the careers' space. There is no utility to return, so `getFiller` (line 2) is not executed.

Input `policy`. The complete list of legal actions is extracted from the policy document.

Output `u`. This is `null`.

Output `actions`. The actions that apply to the active changes are selected from the policy document and returned. For instance, if a set of tags is available, the relevant actions are tailor (**a3**) and display (**a4**). This mapping between changes and actions is a heuristic that is programmed in the analysis module rather than read from the policy document as in the case of SAR.

[9]All software sensors and the accompanying APIs are implemented by David Bennett, Codevate

[10]`https://www.mturk.com/mturk/welcome`

6.2.6 Plan

The logic of the `plan` algorithm is presented in Table 4.4. Here follow the values for the input and output parameters in the CDS problem domain.

Inputs `planBank` and `u`. These are `null`.

Input `actions`. The actions selected during analysis are passed on to the plan module via this input parameter. Method `selectAction` (line 5 in Table 4.4) is in charge of building a plan from the available actions with respect to the reasoner output. For instance, should the candidate actions be edit (**a2**), classify (**a5**), tailor (**a3**) and visualise (**a4**), a possible combination if, in the current CDS state, **M3** and **M2** are active at the same time, **a3** and **a4** will only be included in the plan it the outcome of **a5** is successful.

Input and output `p`. Plan `p` starts empty and is gradually built by including the actions selected during analysis, provided that the consistency of the ontology allows it.

6.3 Implementation Scenario

One step of the KAS methodology (Table 4.5) is illustrated on the CDS scenario in the following. Since this is a finished, web application that is freely available to use by the general public (`http://gcg-test.codevate.com/`), it is useful to understand the information flow between the back-end, semantic-autonomic layer and the front-end, web-based one. This is reflected by the flow diagram in Fig. 6.7[11], where the KAS instance for CDS is depicted in the middle block titled `Ontology server`. The other components (not inlcuded in the KAS arcitecture) are necessary for the system's online operation as described below.

- **OS, OT, OE and OC.** These represent, respectively, the ontology segmenter, tailor, editor and classifier. These are the server-side tools that execute the relevant plan actions on the ontology (or its temporary copy as in the case of the editor).

- **searcher and visualiser.** These are part of the user interface (fig. 6.8) and allow registered members of the public to input search keywords and view the result (namely, the ontology segment containing first and second order parents, children and siblings of the concept matching the search keyword).

- **tagger and the tags collection.** The tagger is a browser add-on (Fig. 6.9 shows it in use - the page being annotated is `nasajobs.nasa.gov`). It is freely available for installation and allows users to tag the web resources they explore with concepts from the ontology. All the annotations used by a given explorer are kept in the tags collection associated to the explorer's account.

- **updater.** Also part of the web interface, this allows registered users to edit the ontology. Fig. 6.10 shows the updater being used to add a child-to-parent connection relationship between `BIOLOGICAL COMPUTING` and `APPLIED BIOLOGICAL SCIENCES`.

---

[11] The search server and the web server are implemented by David Bennett, Codevate

- **DB and search algorithm.** These represent a relational database that stores all ontology concept names and a search algorithm supporting the auto-completion feature provided by the searcher. These are necessary to allow keyword suggestions to be offered in realtime.
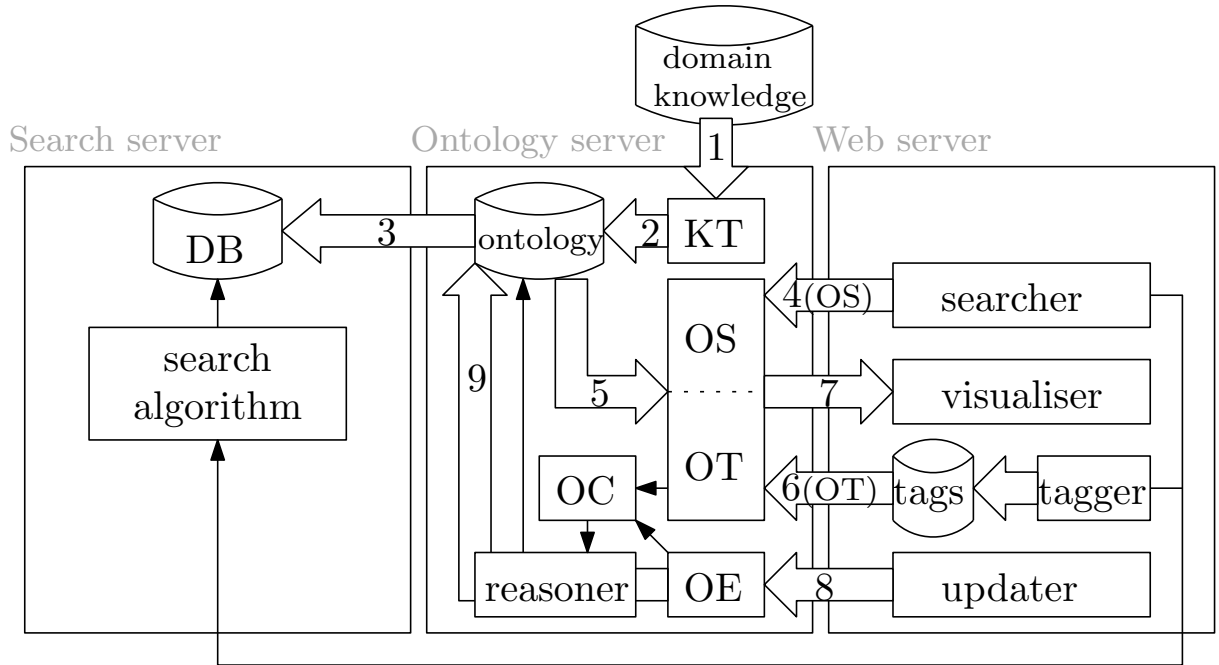

Fig. 6.7: KAS with state-less ontology - component view

6.3.1 Policy Definition and Ontology Learning

In the preparatory stage, method `createPolicy` (line 1 in Table 4.5) will create the policy document with the actions prescribed by either the domain expert or the application developer. The input file is generated (by running `createInputFile` - line 2) with the data compiled by the domain expert from the knowledge sources presented in 6.2.1. With that as input, KT will produce an Onto++ instance without discretisation (no `State` hierarchy), but with reification (supported by the `Link` hierarchy - Fig. 6.2).

6.3.2 Monitoring, Analysis and Planning

Let us assume that **M4**, **M2** and **M3** are active at the same time. This means that the user typed a keyword (for instance, QUANTITATIVE METHODS, as illustrated in Fig. 6.8) in the searcher, a set of tags is avilable to create and display a personal ontology with and some temporary changes have been submitted through the updater (Fig. 6.10). The monitor will send all this information to the analyse module.

During analysis, the following actions will be selected: in reponse to **M3**, edit and classify, in response to **M4**, segment and display, in response to **M2**, tailor and display. The planner (namely lines 11 - 16 in Table 4.5) will consider each of those actions, in turn (only the third and final part of the `while` condition is active as there are no utilities nor a maximum plan length in the CDS domain).
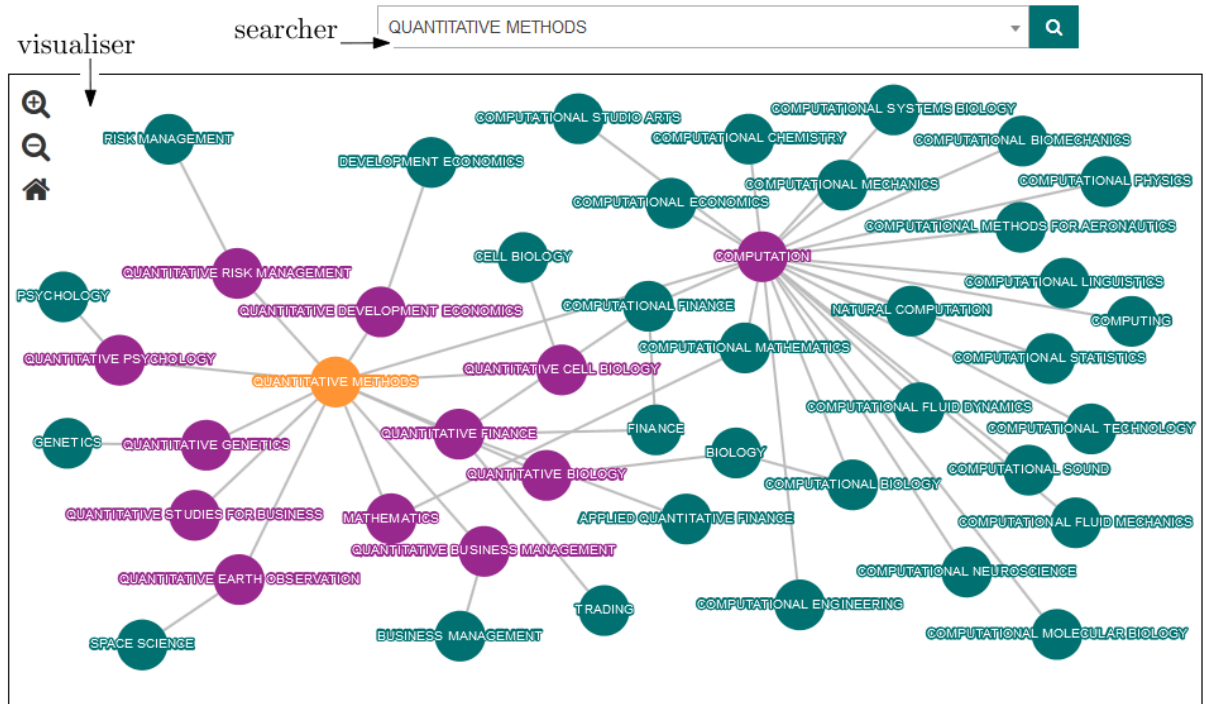
Fig. 6.8: KAS for CDS front-end: searcher and visualiser

Fig. 6.9: KAS for CDS front-end: tagger

Iteration 1. The edit action will be included in the current plan p (line 12). Updates are executed on a temporary copy of the ontology (line 13). There is no reasoner invocation associated to an editing operation, therefore monitoring the reasoner output (line 14) and analysing it (line 15) are both skipped.

Iteration 2. The classify action is added to the plan and executed on the temporary copy of the ontology. This does entail running the reasoner, therefore the answer it provides (whether the edited

Link concepts

Source | BIOLOGICAL COMPUTING | ▼

- BIOLOGY(PARENT)
- BIOINFORMATICS(SIBLING)
- APPLIEDCOMPUTING(PARENT)
- COMPUTER-BASEDBIOLOGY(CHILD)

Destination | APPLIED BIOLOGICAL SCIENCES / APPLIED BIOSCIENCE / APPLIED BIOSCIENCES | ▼

- BIOLOGICALSCIENCES(PARENT)

Type | Child (source is the child of destination) | ▼

% Link
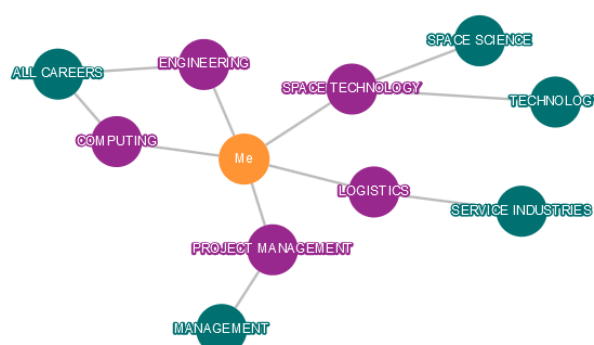
Fig. 6.10: KAS for CDS front-end: updater

Fig. 6.11: KAS for CDS front-end: personal ontology

ontology is consistent or not) will be monitored (line 14) and analysed (line 15) to determine whether to commit the changes to the public ontology.

Iteration 3. After the inclusion of the segment action, that too is executed on the ontology. This is a reasoning-free action, therefore no additional monitoring/ analysis is necessary.

Iteration 4. Display is the next action to include in the plan. This will be executed on the managed resource by populating the display (visualiser in Fig. 6.8) with the graph produced in iteration 3. The displayed graph will either reflect the edits submitted at iteration 1 or not, depending on the outcome of the classification performed in iteration 2. Again, display actions do not require reasoning, so the monitor and analysis steps on lines 14 and 15 are skipped.

Iterations 5 and 6. The final two actions, tailor and display, will be included in plan p and executed without running the reasoner. The outcome of the latter action, namely the personal ontology, is displayed in the visualiser. Fig. 6.11 shows the personal ontology generated for a user (represented by node Me) who annotated webpages with ontology nodes COMPUTING, ENGINEERING, LOGISTICS, SPACE TECHNOLOGY and PROJECT MANAGEMENT. The first order neighbours of those nodes are also

displayed and can be interpreted as suggestions for further exploration.

Afterthoughts about the personal ontology. This personalised graph contains all nodes a given member has used in her online career exploration history, since creating her account. Thus, it can be seen as a "mile marker" representing the user's professional interests at the current moment in time. Companies may also register on the system and create personal ontologies for the candidate they envisage would be ideal to fill a certain role. Interested applicants can evaluate their eligibility by comparing their personal ontology against that of the ideal candidate, noticing the overlaps and discrepancies and ultimately getting an idea of the areas they need to work on in order to become stronger contestants.

6.3.3 Self-management support

The KAS implementation in the CDS domain supports the realisation of the four self-management properties [101].

- **Self-optimisation.** The goal of this KAS instance is to provide its users with complete, correct and intuitive knowledge about the career domain. The system continually optimises its service by improving the "completeness" of the available information, that is, expanding the ontology with new concepts and properties, either explicitly suggested by the users or extracted from their relevant website browsing activity. By allowing/rejecting ontology updates based on their author's credibility (consolidated as the level of interaction with the system increases), the existing model evolves towards accurately representing the community's view of the professional world, rather than the potentially biased understanding of a small group of domain experts. This way, a job seeker will receive better, more complete and more reliable guidance with respect to available roles and the qualifications they require.

- **Self-protection.** User edits causing the ontology to become unsatisfiable, for instance, adding two properties stating that COMPUTING both is and is not a sibling of INFORMATICS, will be flagged by the reasoner during ontology classification and ultimately rejected. Also, the system protects against errors in the input file: if a cycle is detected, the concepts involved will be inferred to be equivalent prompting the ontologist to revisit the source repository.

- **Self-healing.** Semantically incorrect user errors, for instance, asserting that MATHEMATICS and MUSIC are synonyms, cannot be detected by the reasoner, as they are formally valid. However, the KAS instance for the CDS problem domain is capable of healing by delegating the responsibility of evaluating such edits to the community. If the author of the flawed edit is discredited several times by the wider group of curators, the associated reputation score will be affected and future updates will require further verification before being accepted in the live ontology.

- **Self-configuration.** Currently, hardware resources are statically allocated to the search server and the ontology one. A future development may consist in configuring a dynamic algorithm to perform this task.

6.4   Evaluation

6.4.1   Autonomic Manager Evaluation

The instance of the KAS architecture deployed in the CDS application domain achieves the **qualitative indicators** (refer to 2.1.3 for detailed definitions) below.

- **Autonomicity:** level 4 (adaptive). The KAS for CDS instance attains its goal of delivering complete (within the scope of the available knowledge sources), correct and intuitive knowledge that is relevant to the user's career-related interests. Plans are dynamically elaborated using candidate actions selected during analysis. However, level 5 on the autonomicity scale is not achieved, given that sensor and effector APIs still have to be configured by the application developer and are not commissioned proactively on the web services market.

- **Architecture:** flat. There is only one autonomic manager controlling the careers space and the visualiser. As opposed to the SAR problem domain, the manager is deployed from a web server and benefits from a user interface.

- **Adaptation approach:** utility based. Although not explicitly stated, as in the SAR context, the goal state is seen as having the maximum utility in the CDS problem domain. The system will not allow the display of a result that does not integrate data from all available sources, is not verified by the reasoner or is not a graph.

- **Learning:** supported. The KAS instance for CDS expands its knowledge base by accepting user edits as well as relevant information (e.g., job adverts) from third party providers, as explained in 6.2.4. All new data is subsumed by the reasoner in the correct hierarchy and automatically reified.

- **Open:** freely available. All web (e.g., Cytoscape) and console (OWL API) based tools used to build the KAS instance for the CDS problem domain are available under a public licence. The logic behind all components comprised by the KAS instance for CDS is described in full detail (6.2).

- **Evolvable.** The proposed autonomic system is extensible and maintainable. The knowledge base is seamlessly extended by every user exploring new resources, tagging them and performing edits on the existing ontology. At an architectural level, the modular KAS structure, where each MAPE component is underpinned by its own algorithm, is easily maintained, should an upgraded version of a specific tool become available.

- **Robustness.** The ontology is exposed to the public, yet protected against logical inconsistencies by the reasoner. Of course, some errors will escape the reasoner's filter. For instance, an edit asserting `GOOSE` as the child of concept `MATHEMATICS` will initially be accepted but ultimately eliminated by the other members of the community. The goal of the ontology is to capture the *consensus* of the intersted public - this may entail tolerating flawed knowledge in the early stages, however, it is also the driving force that will eliminate inconsistencies in the long term. Given this mecanism, ommunity curated ontologies are implicitly robust.

- **Validation:** mixed. A combination of qualitative and quantitative analysis methods are used to evaluate the proposed KAS instance.

Qualitative evaluation summary. As in the case of SAR, most qualitative indicators are positive. The main difference is observed with respect to openness and scope, as KAS for CDS is intended to cater to a much larger audience and, partly due to the increased computational capabilities of server specific hardware, is capable of managing a significantly larger volume of knowledge. Yet, as will become more evident in the quantitative analysis chapter, some of the subtle mechanics of the KAS architecture that were easy to observe in the SAR context (such as the interplay between environmental change frequency and the runtime duration of the MAPE cycle) will become obscured in the much more complex CDS domain.

The **quantitative measurements** for autonomic behaviour evaluation (2.1.3) that apply to the CDS problem domain are given below.

- **The cost of autonomy** is defined as the runtime duration of the most computationally expensive operations in the MAPE loop: ontology learning and classification. On a Digital Ocean web architecture operating on 4 CPUs @ 8GB RAM for the ontology server and 2 CPUs @ 4GB RAM for the web server, KT takes 7s, whereas classification performed by the FaCT++ reasoner terminates in 140s. The experiments were performed on a spreadsheet (with knowledge collected by the domain expert) of 10000 rows.

- **The speed of the autonomic response** is defined as the average execution time of a query. Semantic querying is the most computationally expensive operation within the autonomic loop, therefore it provides a good approximation for the MAPE cycle duration. The query durations (averaged over a series of 10 experiments) for a target ontology node with less than 50 neighbours, between 50 and 100 neighbours, and over 100 neighbours are presented in Table 6.2. The same web server configuration was used as for the cost of autonomy. The target ontology node refers to the concept that matches the user's search keyword. The number of neighbours includes first and second order relations of the target ontology node (namely concepts separated from the target node by at most two sibling, parent or child links).

- **The learning index** - the proposed way to calculate this is by using the same formula as the one suggested for SAR (equation 5.1). In the CDS domain, $p+$ would represent the number of initially tolerated bad edits (such as the previous GOOSE example) and $p-$ would stands for the number of ultimately rejected bad entries. Unfortunately, in the short time since the system prototype has been released online (summer 2015), not enough data has been collected to allow such a calculation, therefore system robustness analysis by means of l_i calculations is planned for future work.

Table 6.2: KAS for CDS - speed of autonomic response evaluation

connectivity	δt[ms]
<50	354
50 - 100	486
>100	2869

Quantitative evaluation summary. The experimental measurements taken in the online operating environment of KAS for CDS reveal that the system meets the computationally efficiency standards of a live deployment. The data in Table 6.2 shows that the support provided by the semantic level (especially the reasoner) to the autonomic components (analysis and planning, in particular) does not impair realtime operation. In perspective, the proposed learning index analysis has the potential of providing more insight into the system's capacity of *self-managing* the ontology.

6.4.2 Semantic Components Evaluation

The qualitative and quantitative metrics available in the ontology related literature (covered in 2.2.5) and applicable to the CDS domain are discussed in the following.

Qualitative Evaluation

ONTOMETRIC. As opposed to the Onto++ instance constructed for the SAR domain, the CDS ontology does support n-ary relationships. Thus, one of the two flexibility criteria proposed by ONTOMETRIC is met. The other, support for concept instances, is addressed in a different way, as explained in the case of SAR: there are no individuals per se in the CDS ontology, as they are modelled with classes, to enable more powerful reasoning.

OOPS! Given the size of the initial CDS ontology, only a segment of it (namely, the one produced in response to a user query) was tested in OOPS!. The provided diagnosis showed no design pitfalls in the considered ontology excerpt. It is noteworthy that this result was obtained from the first try (no KT amendments were necessary as in the SAR case - 5.4.2). Thus, the improvements operated on KT during SAR evaluation increased the tool's efficiency across problem domains.

Quantitative Evaluation

Temporal bias cannot be measured by means of cosine similarity, as suggested in [79], given the extremely high number of dimensions that the careers vector space would have. However, the CDS ontology does remain an accurate representation of the modelled domain, given the way it supports learning: as long as users continue to tag new relevant web resources with ontology concepts and also edit the ontology in light of their newly acquired expertise, temporal bias will remain small.

Evaluating the cost of the ontology engineering process would be a complicated process, given the multitude of knowledge sources and the development effort involved in each OE stage described in 6.2.1. However, that is a one-off cost as, once the initial version of the ontology is extracted from the domain expert's spreadsheet, the ontology life cycle becomes partly *automated* - supported by the autonomic loop - and partly *delegated* to explorers (Amazon's mechanical turk scenario).

Ontology cohesion [68] is a quantitative measure that did not apply to SAR but is relevant to CDS. In its original interpretation, this is calculated as

$$1 - \frac{orphans}{hubs},\tag{6.1}$$

where *orphans* refers to completely isolated ontology nodes and *hubs* represents the number of very well connected concepts. In the CDS initial ontology, as extracted by KT from the input file, cohesion

would always be 1, since isolated nodes are implicitly eliminated during the OL process. A more relevant cohesion indicator would be to replace the number of orphans in equation 6.1 with that of poorly connected nodes (less than 50 first and second order relations). A simple Java application was developed to count poorly and highly connected nodes. It found that approximately 20% of the initial ontology version had less than 50 relations, whereas 23% had more than 100. This accounts for a poor-hub ratio of around 0.87, leading to a cohesion value of 0.13. This relatively low cohesion is to be expected in the initial stages of the ontology life cycle, as it indicates that several areas of the careers' space have not yet been sufficiently explored. As more data from the online operation of the CDS system becomes available, it would be useful to investigate how this cohesion score evolves over time.

Summary

As in the case of SAR, the CDS ontology analysis indicates a good compliance with recommended design patterns and supports the sustainability of following an OE methodology such as NeOn. As a special note, further analysis of ontology cohesion evolution will be performed in the future, as more data becomes available.

Conclusions

7.1 Contributions Overview

The presented work proposes a flexible *architecture*, the accompanying *toolset* and an implementation *methodology* for an autonomic IT infrastructure employing a semantic knowledge management platform. The ensemble of these three elements is titled KAS (Knowledge-centric Autonomic System). In this context, the main contributions target the following areas:

- **Design.** The architecture underpinning KAS was developed as a blueprint, describing a flexible configuration of autonomic elements (MAPE modules) and semantic components (ontology and reasoner). The aim is for the architecture to be *concrete* enough to allow specific instances to be built by following its design yet, at the same time, *flexible* and *configurable*, to enable usage across multiple application domains.

- **Implementation.** Supporting tools for each architecture component were described both in isolation and in interaction (by means of a generic methodology to control their execution flow). The role of the proposed tools within the KAS architecture was illustrated and analysed on specific implementation scenarios.

- **Application.** To demonstrate the adaptability of KAS to various practical scenarios, the architecture, toolset and methodology were instantiated on two separate application domains, namely a self-adaptive document rendering (SAR) problem and a career decision support (CDS) platform. The instances exploit different parts of the architectural template and provide distinct interpretations for some of the methodology, yet they both clearly implement KAS.

- **Evaluation.** A set of qualitative and quantitative metrics was extracted from the relevant literature to create a common evaluation framework for the two KAS instances. The metrics were either applied directly or adapted, in order to investigate the impact of semantic reasoning on autonomic behaviour. This analysis provided insight into the relationship between domain representation accuracy (state models, reified properties, etc.) and learning (as supported by the reasoner), on the one hand, and autonomic planning and analysis, on the other hand.

- **Reflection.** Several design approaches and alternative tool implementations were considered before making the final selection. This provided the context for critically analysing existing similar solutions and distilling a set of guidelines and design principles for improving them through KAS.

The success of that effort was assessed by interpreting the qualitative and numerical outcomes of the proposed experimental case studies.

7.2 Lessons Learnt

The experience of framework development, implementation, evaluation and practical results analysis has yielded the following conclusions.

- Semantic reasoning efficiency increases when the ontology it is performed on is developed in compliance with a robust OE methodology (NeOn was chosen for both KAS instances). This is chiefly due to the positive impact that observing good practices and design patterns while extracting the ontology have on learning, a powerful facilitator of autonomic behaviour. This conclusion was reached while addressing the first research question (1.2), relevant to the design of an effective semantic-autonomic hybrid architecture.

- The modularity (low coupling, high cohesion) of the KAS architecture and methodology allows for tools to be replaced with upgraded versions, for heuristics to be redefined, swapped, extended, etc., without significantly impacting the implementation of the other architectural components. This insight was gained in connection to investigating the second research question, pertaining to the practical implementation of the proposed KAS architecture.

- The flexibility of semantic platforms is to be preferred over more traditional knowledge management solutions in situations where the modelled concepts form a natural hierarchy (there is no native support for inheritance in relational databases [114]). Other contextual factors in favour of ontology use over alternative models are the need for insightful, contextual query results, a higher emphasis on learning than on speed, continuously changing domains that feature uncertainties, etc. This conclusion also relates to architectural design, therefore is relevant to the first research question.

- Natural language processing is expensive and experimental. However, some form of automated translation is necessary to convert the natural language that domain experts/ end users communicate in into the machine readable formalism that autonomic systems employ. An intuitive, cost efficient way of doing that is exposing the manager's knowledge to the human users in the form of a graph (especially since that is intrinsically compatible with the structure of ontology concept and property models). The KAS instance for the CDS problem domain uses ontology visualisation in the form of a graph as a means to provide perspective on a domain that is otherwise convoluted and heterogeneous and where subtle connections are difficult to observe. Endowing the KAS implementation in the career domain with a visualisation tool is connected to the second research question.

7.3 Research Dissemination

The work presented in this thesis is the subject of several publications:

- The preliminary analysis of the autonomic-semantic interaction and its relevance to other related fields, such as semantic web services, was included in a broader scope survey paper.

  Wang, H. H., Gibbins, N., Payne, T., **Patelli, A.**, & Wang, Y. (2015). A survey of semantic web services formalisms. Concurrency and Computation: Practice and Experience, 27(15), 4053-4072.

- The research rationale and framework in the SAR domain was discussed in:

  Patelli, A., Calinescu, R., & Wang, H. (2014, June). Semantic reasoning for autonomic IT systems. In Proceedings of the 19th international doctoral symposium on Components and architecture (pp. 13-18). ACM.

- The practical features of the KAS instance for the CDS problem domain were discussed in

  Patelli, A., Lewis, P., Wang, H., Nabney I., Bennett D. & Lucas R. (2016). GCG Aviator: A Decision Support Agent for Career Management

  a demo paper accepted for publication by the Artificial Life and Intelligent Agents International Symposium.

- The architecture and knowledge space curation application of the KAS instance for the CDS problem domain were analysed in

  Patelli, A., Lewis, P., Wang, H., Nabney I., Bennett D., Lucas R. & Cole A. (2016). Autonomic Curation of Crowdsourced Knowledge: The Case of Career Data Management

  accepted for publication by the International Conference on Cloud and Autonomic Computing.

7.4 Future Work

The main identified avenues for future research are:

- The development of an intermediate layer in the KAS architecture to allow interpreting online descriptions of semantic web services, selecting the ones capable of configuring the APIs for the employed sensors and executing them, thus enabling the system to receive monitored data without the application developer's help. This would support the increase of KAS instances' autonomicity level for 4 to 5.

- Investigation into the architectural variation where policies and plans are included in the system ontology (and therefore reasoned upon).

- Automation of the mapping process between the personal ontology of KAS for CDS users and that published by recruiting companies to describe the profile of an ideal candidate.

- Additional data collection from the online operation of the KAS for CDS instance to support further experimental analysis based on the learning index and the ontology cohesion measures.

Bibliography

[1] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Future Gener. Comput. Syst.*, 18(8):1061–1074, 2002.

[2] S. Agarwala, Y. Chen, D. S. Milojicic, and K. Schwan. QMON: QoS- and utility-aware monitoring in enterprise systems. In *ICAC*, pages 124–133, 2006.

[3] S. Agrawal, S. Chaudhuri, and V. Narasayya. Materialized view and index selection tool for microsoft sql server 2000. *ACM SIGMOD Record*, 30(2):608, 2001.

[4] Y. W. Ahn and A. M. K. Cheng. Autonomic computing architecture for real-time medical application running on virtual private cloud infrastructures. *ACM SIGBED Review*, 10(2):15–15, 2013.

[5] K. Ahuja and H. Dangey. Autonomic computing: An emerging perspective and issues. In *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*, pages 471–475. IEEE, 2014.

[6] M. M. Al-Zawi, D. Al-Jumeily, A. Hussain, and A. Taleb-Bendiab. Autonomic computing: Applications of self-healing systems. In *Developments in E-systems Engineering (DeSE), 2011*, pages 381–386. IEEE, 2011.

[7] M. B. Alaya and T. Monteil. Frameself: an ontology-based framework for the self-management of machine-to-machine systems. *Concurrency and Computation: Practice and Experience*, 27(6):1412–1426, 2015.

[8] P. Alexopoulos, J. Pavlopoulos, M. Wallace, and K. Kafentzis. Exploiting ontological relations for automatic semantic tag recommendation. In *Proceedings of the Seventh International Conference on Semantic Systems*, pages 105–110, New York, NY, USA, 2011. ACM.

[9] J. Alonso. *OASys: Ontology for Autonomous Systems*. PhD thesis, Departamento de Automatica, Ingenieria Electronica e Informatica Industrial, Universidad Politecnica de Madrid, 2010.

[10] G. Antoniou and F. Van Harmelen. Web ontology language: Owl. In *Handbook on ontologies*, pages 67–92. Springer, 2004.

[11] M. S. Aphale, T. J. Norman, and M. Sensoy. Goal directed conflict resolution and policy refinement. 2012.

[12] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger. Oceano-sla based management of a computing utility. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 855–868. IEEE, 2001.

[13] M. D. Aquin, E. Motta, M. Sabou, S. Angeletou, L. Gridinoc, V. Lopez, and D. Guidi. Toward a new generation of semantic web applications. *Intelligent Systems, IEEE*, 23(3):20–28, 2008.

[14] H. Arora, T. Raghu, A. Vinze, and P. Brittenham. Collaborative self-configuration and learning in autonomic computing systems: Applications to supply chain. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 303–304. IEEE, 2006.

[15] D. Artz and Y. Gil. A survey of trust in computer science and the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):58–71, 2007.

[16] K. Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7):97–114, 2009.

[17] M. Assel, A. Cheptsov, G. Gallizo, I. Celino, D. Dell'Aglio, L. Bradeško, M. Witbrock, and E. D. Valle. Large knowledge collider: a service-oriented platform for large-scale semantic reasoning. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 41. ACM, 2011.

[18] N. Ayari, A. Chibani, Y. Amirat, and E. T. Matson. A novel approach based on commonsense knowledge representation and reasoning in open world for intelligent ambient assisted living services. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 6007–6013. IEEE, 2015.

[19] F. Baader, D. Calvanese, D. McGuiness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[20] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 15–22. IEEE, 2002.

[21] A. Bassi, S. Denazis, A. Galis, C. Fahy, M. Serrano, and J. Serrat. Autonomic internet: a perspective for future internet services based on autonomic principles. *Management of Networks and Services Manweek*, 2007.

[22] V. R. Benjamins. Near-term prospects for semantic technologies. *Intelligent Systems, IEEE*, 23(1):76–88, 2008.

[23] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):28–37, 2001.

[24] J. Bigus, D. Schlosnagle, J. R. Pilgrim, W. Mills III, and Y. Diao. Able: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, 41(3):350–371, 2002.

[25] E. Blomqvist, A. Gangemi, and V. Presutti. Experiments on pattern-based ontology design. In *Proceedings of the fifth international conference on Knowledge capture*, pages 41–48. ACM, 2009.

[26] O. Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32(suppl 1):D267–D270, 2004.

[27] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May. The autonomic network architecture (ana). *Selected Areas in Communications*, 28(1):4–14, 2010.

[28] J. M. Bradshaw, A. Uszok, M. Breedy, L. Bunch, T. Eskridge, P. Feltovich, M. Johnson, J. Lott, and M. Vignati. The kaos policy services framework. In *8th Cyber Security and Information Intelligence Research Workshop*, 2013.

[29] K. Breitman, C. H. Felicissimo, and W. Truszkowski. The autonomic semantic desktop: helping users cope with information system complexity. In *Engineering of Autonomic and Autonomous Systems, 2006. EASe 2006. Proceedings of the Third IEEE International Workshop on*, pages 158–164. IEEE, 2006.

[30] M. Buffa, F. Gandon, G. Ereteo, P. Sander, and C. Faron. Sweetwiki: A semantic wiki. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):84–97, 2008.

[31] A. Calì, D. Calvanese, S. Colucci, and T. D. N. F. M. Donini. A description logic based approach for matching user profiles. In *2004 International Workshop on Description Logics*, page 110, 2004.

[32] R. Calinescu. General-purpose autonomic computing. In M. Denko et al., editors, *Autonomic Computing and Networking*, pages 3–20. Springer, 2009.

[33] R. Calinescu, S. Kikuchi, and M. Kwiatkowska. Formal methods for the development and verification of autonomic IT systems. In P. Cong-Vinh, editor, *Formal and Practical Aspects of Aut. Comp. and Networking*, pages 1–37. IGI Global, 2010.

[34] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic it systems. In *Proceedings of the 31st International Conference on Software Engineering*, pages 100–110. IEEE Computer Society, 2009.

[35] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83. ACM, 2004.

[36] R. Carroll, J. Strassner, G. Cox, and S. van der Meer. Policy and profile: Enabling self-knowledge for autonomic systems. In *Large Scale Management of Distributed Systems*. Springer, 2006.

[37] R. Chadha, H. Cheng, Y.-H. Cheng, J. Chiang, A. Ghetie, G. Levin, and H. Tanna. Policy-based mobile ad hoc network management. In *Policies for Distributed Systems and Networks, 2004*, pages 35–44. IEEE, 2004.

[38] P. Charlton and G. Magoulas. Autonomic computing and ontologies to enable context-aware learning design. In *ICTAI*, pages 286–291, 2010.

[39] S.-W. Cheng, D. Garlan, and B. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pages 2–8. ACM, 2006.

[40] A. Cheptsov and Z. Huang. Making web-scale semantic reasoning more service-oriented: The large knowledge collider. In *Web Information Systems Engineering–WISE 2011 and 2012 Workshops*, pages 13–26. Springer, 2013.

[41] S. K. Chilukuri and K. Doraisamy. Symptom database builder for autonomic computing. In *Autonomic and Autonomous Systems, 2006. ICAS'06. 2006 International Conference on*, pages 32–32. IEEE, 2006.

[42] P. Cimiano and J. Völker. Text2onto. In *Natural language processing and information systems*, pages 227–238. Springer, 2005.

[43] P. Clark and P. Harrison. Boeing's nlp system and the challenges of semantic representation. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 263–276. Association for Computational Linguistics, 2008.

[44] O. Corby, R. Dieng-Kuntz, F. Gandon, and C. Faron-Zucker. Searching the semantic web: Approximate query processing based on ontologies. *Intelligent Systems, IEEE*, 21(1):20–27, 2006.

[45] R. Cuel, A. Delteil, V. Louis, and C. Rizzi. Knowledge web technology roadmap" the technology roadmap of the semantic web. *Knowledge Web*, 2004.

[46] H. Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[47] C. d'Amato, V. Bryl, and L. Serafini. Semantic knowledge discovery from heterogeneous data sources. In *Knowledge Engineering and Knowledge Management*, pages 26–31. Springer, 2012.

[48] M. d'Aquin and E. Motta. Watson, more than a semantic web search engine. *Semantic Web*, 2(1):55–63, 2011.

[49] M. d'Aquin and N. F. Noy. Where to publish and find ontologies? a survey of ontology libraries. *Web Semantics: Science, Services and Agents on the World Wide Web*, 11(0):96 – 111, 2012.

[50] E. M. Dashofy, A. Van der Hoek, and R. N. Taylor. Towards architecture-based self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 21–26, 2002.

[51] C. J. Date. *The Database Relational Model: A Retrospective Review and Analysis: A Historical Account and Assessment of EF Codd's Contribution to the Field of Database Technology*. Addison Wesley, 2001.

[52] M. T. Diallo, H. Moustafa, H. Afifi, and N. Marechal. Adaptation of audiovisual contents and their delivery means. *Commun. ACM*, 56(11):86–93, 2013.

[53] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

[54] Y. Ding, E. Jacob, M. Fried, I. Toma, E. Yan, and S. Foo. Upper tag ontology (UTO) for integrating social tagging data. *Journal of the American Society for Information Science and Technology*, 61(3):505–521, 2010.

[55] C. Diop, E. Exposito, C. Chassot, and D. Jlidi. QoS-Aware and Ontology-Driven Autonomic Service Bus. In *WETICE*, pages 417–422, 2012.

[56] S. Dobson, L. Coyle, and P. Nixon. Hybridizing events and knowledge as a basis for building autonomic systems. *IEEE TCAAS Letters*, 2007.

[57] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao. Autonomia: an autonomic computing environment. In *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, pages 61–68. IEEE, 2003.

[58] M. Dudáš, O. Zamazal, and V. Svátek. Roadmapping and navigating in the ontology visualization landscape. In *Knowledge Engineering and Knowledge Management*, pages 137–152. Springer, 2014.

[59] M. Dzbor, E. Motta, and J. Domingue. Magpie: experiences in supporting semantic web browsing. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(3):204–222, 2007.

[60] D. Ejigu, M. Scuturici, and L. Brunie. An ontology-based approach to context modeling and reasoning in pervasive computing. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference On*, pages 14–19. IEEE, 2007.

[61] L. Fallon and D. O'sullivan. Aesop: A semantic system for autonomic management of end-user service quality. In *IM*, pages 716–719, 2013.

[62] L. L. Fong, M. Kalantar, D. P. Pazel, G. S. Goldszmidt, S. Fakhouri, and S. Krishnakumar. Dynamic resource management in an eutility. In *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, pages 727–740. IEEE, 2002.

[63] B. Fu, N. F. Noy, and M.-A. Storey. Indented tree or graph? a usability study of ontology visualization techniques in the context of class mapping evaluation. In *The Semantic Web–ISWC 2013*, pages 117–134. Springer, 2013.

[64] B. Furletti and F. Turini. Knowledge discovery in ontologies. *Intelligent Data Analysis*, 16(3):513–534, 2012.

[65] A. Gangemi and V. Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009.

[66] A. Gangemi, M.-T. Sagri, and D. Tiscornia. A constructive framework for legal ontologies. In *Law and the semantic web*, pages 97–124. Springer, 2005.

[67] R. García-Castro, A. Gómez-Pérez, and O. Munoz-Garcia. The semantic web framework: a component-based framework for the development of semantic web applications. In *Database and Expert Systems Application*, pages 185–189. IEEE, 2008.

[68] F. García-Peñalvo, J. García, and R. Therón. Analysis of the owl ontologies: A survey. *Scientific Research and Essays*, 6(20):4318–4329, 2011.

[69] N. George, R. Jasper, M. R. LaFever, K. Morrison, D. Rosenthal, S. Tockey, J. Woolley, J. Bradshaw, G. Boy, and P. Holm. Kaos: A knowledgeable-agent-oriented system. In *AAAI Spring Symposium on Software Agents*, pages 21–23, 1994.

[70] G. Gharbi, M. Ben Alaya, C. Diop, and E. Exposito. AODA: An Autonomic and Ontology-Driven Architecture for Service-Oriented and Event-Driven Systems. In *WETICE*, pages 72–77, 2012.

[71] C. Ghezzi, A. Mocci, and M. Monga. Synthesizing intensional behavior models by graph transformation. In *Proc. of the 31st Int. Conf. on Software Engineering*, pages 430–440, 2009.

[72] A. Gomez-Perez and D. Manzano-Mancho. A survey of ontology learning methods and techniques. Deliverable 1.5, OntoWeb Consortium, 2003.

[73] B. Good, E. Kawas, and M. Wilkinson. Bridging the gap between social tagging and semantic annotation: E.D. the entity describer. *Nature Precedings*, 2007.

[74] S. Götz, J. Mendez, V. Thost, and A.-Y. Turhan. Owl 2 reasoning to detect energy-efficient software variants from context. In *OWLED*. Citeseer, 2013.

[75] L. Grunske. Specification patterns for probabilistic quality properties. In *ICSE'08*, pages 31–40. IEEE, 2008.

[76] H.-J. Happel and S. Seedorf. Applications of ontologies in software engineering. In *semantic web enabled software engineering*, pages 1–14. Citeseer, 2006.

[77] M. Hepp. Possible ontologies: How reality constrains the development of relevant ontologies. *Internet Computing, IEEE*, 11(1):90–96, 2007.

[78] M. Hepp, D. Bachlechner, and K. Siorpaes. Harvesting wiki consensus - using wikipedia entries as ontology elements. In *Semantic Wikis*, pages 124–139. ESWC06, 2006.

[79] H. Hloman and D. A. Stacey. Multiple dimensions to data-driven ontology evaluation. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 329–346. Springer, 2014.

[80] V. Holub, T. Parsons, P. O'Sullivan, and J. Murphy. Run-time correlation engine for system monitoring and testing. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, pages 9–18. ACM, 2009.

[81] P. Horn. Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM, 2001.

[82] M. Horridge. A practical guide to building owl ontologies using protégé 4 and co-ode tools edition 1.3. *The University of Manchester*, 2011.

[83] I. Horrocks. Ontologies and the semantic web. *Communications of the ACM*, 51(12):58–67, 2008.

[84] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic web architecture: Stack or two towers? In *Principles and practice of semantic web reasoning*, pages 37–41. Springer, 2005.

[85] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21:79, 2004.

[86] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7–26, 2003.

[87] H. Hu, G.-J. Ahn, and K. Kulkarni. Ontology-based policy anomaly management for autonomic computing. In *Collaborative Computing: Networking, Applications and Worksharing*, pages 487–494. IEEE, 2011.

[88] M. C. Huebscher and J. A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 40(3):7:1–7:28, 2008.

[89] M. C. Huebscher, J. A. McCann, and A. Hoskins. Context as autonomic intelligence in a ubiquitous computing environment. *Internet Protocol Technology*, 2(1):30–39, 2006.

[90] IBM. An architectural blueprint for autonomic computing. Technical report, IBM, 2005.

[91] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin. *A practical guide to the IBM autonomic computing toolkit*. IBM, International Technical Support Organization, 2004.

[92] B. Jafarpour, S. R. Abidi, and S. S. R. Abidi. Exploiting owl reasoning services to execute ontologically-modeled clinical practice guidelines. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 307–311. Springer, 2011.

[93] V. Janev and S. Vraneš. Applicability assessment of semantic web technologies. *Information Processing & Management*, 47(4):507–517, 2011.

[94] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *POLICY*, pages 63–74. IEEE, 2003.

[95] G. Kaiser, J. Parekh, P. Gross, and G. Valetto. Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In *Autonomic Computing Workshop*, pages 22–30, 2003.

[96] Y.-B. Kang, Y.-F. Li, and S. Krishnaswamy. Predicting reasoning performance using ontology metrics. In *The Semantic Web–ISWC 2012*, pages 198–214. Springer, 2012.

[97] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology visualization methods—a survey. *ACM Computing Surveys*, 39(4):10, 2007.

[98] A. Kato, S. Tsuchiya, and M. Adachi. TRIOLE organic computing architecture. *FUJITUS Sci. Tech. J.*, 43(4):412–419, October 2007.

[99] E. Kaufmann, A. Bernstein, and L. Fischer. Nlp-reduce: A "naïve" but domain-independent natural language interface for querying ontologies. ESWC Zurich, 2007.

[100] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*, pages 68–78. IEEE, 1998.

[101] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[102] J. Kephart and W. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Policies for Distributed Systems and Networks*, pages 3–12, 2004.

[103] J. O. Kephart. Research challenges of autonomic computing. In *Proceedings of the 27th international conference on Software engineering*, pages 15–22. ACM, 2005.

[104] A. Khalid, M. A. Haye, M. J. Khan, and S. Shamail. Survey of frameworks, architectures and techniques in autonomic computing. In *Autonomic and Autonomous Systems, 2009. ICAS'09. Fifth International Conference on*, pages 220–225. IEEE, 2009.

[105] E. Kiciman and Y.-M. Wang. Discovering correctness constraints for self-management of system configuration. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 28–35. IEEE, 2004.

[106] M. Knorr, P. Hitzler, and F. Maier. Reconciling owl and non-monotonic rules for the semantic web. 2012.

[107] I. Kollia, B. Glimm, and I. Horrocks. Sparql query answering over owl ontologies. In *The Semantic Web: Research and Applications*, pages 382–396. Springer, 2011.

[108] A. A. Kopiler, I. D. C. Dutra, and F. M. G. França. Personal autonomic desktop manager with a circulatory computing approach. In *Engineering of Autonomic and Autonomous Systems*, pages 119–127. IEEE, 2008.

[109] R. Kowalski. The logical way to be artificially intelligent. In *Computational Logic in Multi-Agent Systems*, pages 1–22. Springer, 2005.

[110] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *Future of Software Engineering, 2007. FOSE '07*, pages 259–268, 2007.

[111] K. Kravari, C. Papatheodorou, G. Antoniou, and N. Bassiliades. Reasoning and proofing services for semantic web agents. In *IJCAI*, volume 2011, pages 2662–2667. Citeseer, 2011.

[112] C. Krupitzer, F. M. Roth, S. Vansyckel, and C. Becker. Towards reusability in autonomic computing. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pages 115–120. IEEE, 2015.

[113] P. Lalanda, J. A. McCann, and A. Diaconescu. *Autonomic Computing: Principles, design and implementation*. Springer Science & Business Media, 2013.

[114] N. Lammari, I. Comyn-Wattiau, and J. Akoka. Extracting generalization hierarchies from relational databases: A reverse engineering approach. *Data & Knowledge Engineering*, 63(2):568–589, 2007.

[115] G. Lanfranchi, P. D. Peruta, A. Perrone, and D. Calvanese. Toward a new landscape of systems management in an autonomic computing environment. *IBM Systems journal*, 42(1):119–128, 2003.

[116] D. Lewis, K. Feeney, K. Carey, T. Tiropanis, and S. Courtenage. Semantic-based policy engineering for autonomic systems. In M. Smirnov, editor, *Autonomic Communication*, volume 3457 of *LNCS*, pages 152–164. Springer Berlin Heidelberg, 2005.

[117] H. Liu and M. Parashar. Accord: a programming framework for autonomic applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3):341–352, 2006.

[118] D. Lizcano, F. Alonso, J. Soriano, and G. López. Automated end user-centred adaptation of web components through automated description logic-based reasoning. *Information and Software Technology*, 57:446–462, 2015.

[119] G. M. Lohman and S. S. Lightstone. Smart: making db2 (more) autonomic. In *Very Large Data Bases*, pages 877–879. VLDB Endowment, 2002.

[120] S. Lohmann, S. Negru, F. Haag, and T. Ertl. Vowl 2: user-oriented visualization of ontologies. In *Knowledge Engineering and Knowledge Management*, pages 266–281. Springer, 2014.

[121] V. Lopez, V. Uren, M. Sabou, and E. Motta. Is question answering fit for the semantic web?: a survey. *Semantic Web*, 2(2):125–155, 2011.

[122] H. Louafi, S. Coulombe, and U. Chandra. Efficient near-optimal dynamic content adaptation applied to jpeg slides presentations in mobile web conferencing. In *Proc. of AINA*, pages 724–731, 2013.

[123] A. Lozano-Tello and A. Gómez-Pérez. Ontometric: A method to choose the appropriate ontology. *Journal of database management*, 2(15):1–18, 2004.

[124] D. F. Macedo, Z. Movahedi, J. Rubio-Loyola, A. Astorga, G. Koumoutsos, and G. Pujolle. The autoi approach for the orchestration of autonomic networks. *annals of telecommunications*, 66(3-4):243–255, 2011.

[125] V. Markl, G. M. Lohman, and V. Raman. Leo: An autonomic query optimizer for db2. *IBM Systems Journal*, 42(1):98–106, 2003.

[126] C. Martinez-Cruz, I. J. Blanco, and M. A. Vila. Ontologies versus relational databases: are they so different? a comparison. *Artificial Intelligence Review*, 38(4):271–290, 2012.

[127] J. A. McCann and M. C. Huebscher. Evaluation issues in autonomic computing. In *Grid and Cooperative Computing-GCC 2004 Workshops*, pages 597–608. Springer, 2004.

[128] B. Melcher and B. Mitchell. Towards an autonomic framework: Self-configuring network services and developing autonomic applications. *Intel Technology Journal*, 8(4):279 – 290, 2004.

[129] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between owl and relational databases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):74–89, 2009.

[130] E. Motta, P. Mulholland, S. Peroni, M. d'Aquin, J. M. Gomez-Perez, V. Mendez, and F. Zablith. A novel approach to visualizing and navigating ontologies. In *The Semantic Web–ISWC 2011*, pages 470–486. Springer, 2011.

[131] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle. A survey of autonomic network architectures and evaluation criteria. *Communications Surveys & Tutorials, IEEE*, 14(2):464–490, 2012.

[132] S. Nepal, C. Paris, and A. Bouguettaya. Trusting the social web: issues and challenges. *World Wide Web*, 18(1):1–7, 2015.

[133] F. Neuhaus, A. Vizedom, K. Baclawski, M. Bennett, M. Dean, M. Denny, M. Grüninger, A. Hashemi, T. Longstreth, L. Obrst, et al. Towards ontology evaluation across the life cycle. *Applied Ontology*, 8(3):179–194, 2013.

[134] B. Neumayr, C. G. Schuetz, and M. Schrefl. Towards ontology-driven rdf analytics. In *Advances in Conceptual Modeling*, pages 210–219. Springer, 2015.

[135] N. Noy, T. Tudorache, C. Nyulas, and M. Musen. The ontology life cycle: Integrated tools for editing, publishing, peer review, and evolution of ontologies. In *AMIA Annu Symp Proc*, volume 2010, pages 552–556. Citeseer, 2010.

[136] N. F. Noy, D. L. McGuinness, et al. Ontology development 101: A guide to creating your first ontology. Technical report, 2001.

[137] N. F. Noy, M. A. Musen, et al. Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00). Available as SMI technical report SMI-2000-0831*, 2000.

[138] J. Z. Pan, S. Staab, U. Aßmann, J. Ebert, and Y. Zhao. *Ontology-driven software development*. Springer Science & Business Media, 2012.

[139] J. Z. Pan and Y. Zhao. *Semantic Web Enabled Software Engineering*, volume 17. IOS Press, 2014.

[140] R. Peinl. Semantic web: State of the art and adoption in corporations. *KI-Künstliche Intelligenz*, pages 1–8, 2016.

[141] T. Pellegrini, A. Blumauer, G. Granitzer, A. Paschke, and M. Luczak-Rösch. Semantic web awareness barometer 2009-comparing research-and application-oriented approaches to social software and the semantic web. In *I-SEMANTICS*, pages 518–529, 2009.

[142] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

[143] G. Petasis, V. Karkaletsis, G. Paliouras, A. Krithara, and E. Zavitsanos. Ontology population and enrichment: State of the art. In *Knowledge-driven multimedia information extraction and ontology evolution*, pages 134–166. Springer-Verlag, 2011.

[144] S. Pigeon and S. Coulombe. Quality-aware predictor-based adaptation of still images for the multimedia messaging service. *Multimedia Tools and Applications*, pages 1–25, 2013.

[145] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa. Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):7–34, 2014.

[146] D. Rajpathak and E. Motta. An ontological formalization of the planning task. In *International Conference on Formal Ontology in Information Systems (FOIS 2004)*, pages 305–316, 2004.

[147] A. Ranganathan and R. H. Campbell. Autonomic pervasive computing based on planning. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 80–87. IEEE, 2004.

[148] D. Raychaudhuri, N. B. Mandayam, J. B. Evans, B. J. Ewy, S. Seshan, and P. Steenkiste. Cognet: an architectural foundation for experimental cognitive radio networks within the future internet. In *Mobility in the evolving internet architecture*, pages 11–16. ACM, 2006.

[149] V. Ricquebourg, D. Durand, D. Menga, B. Marine, L. Delahoche, C. Loge, and A.-M. Jolly-Desodt. Context inferring in the smart home: An swrl approach. In *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, volume 2, pages 290–295. IEEE, 2007.

[150] M. Salehie and L. Tahvildari. Autonomic computing: Emerging trends and open problems. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005.

[151] S. Schlobach, R. Cornet, et al. Non-standard reasoning services for the debugging of description logic terminologies. In *IJCAI*, volume 3, pages 355–362, 2003.

[152] B. Schmerl and D. Garlan. Acmestudio: Supporting style-centered architecture development. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 704–705. IEEE, 2004.

[153] P. Schmitz. Inducing ontology from flickr tags. In *Proceedings of the Collaborative Web Tagging Workshop*, 2006.

[154] J. Schütte and T. Wahl. A description logic based approach on handling inter-domain policy conflicts using meta-policies. *IEEE Vehicular Technology Magazine*, 2010.

[155] M. Sensoy, T. J. Norman, W. W. Vasconcelos, and K. Sycara. Owl-polar: A framework for semantic policy representation and reasoning. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12:148–160, 2012.

[156] N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, et al. Cs aktive space, or how we learned to stop worrying and love the semantic web. *IEEE Intelligent Systems*, (3):41–47, 2004.

[157] E. Simperl, T. Bürger, S. Hangl, S. Wörgl, and I. Popov. Ontocom: A reliable cost estimation method for ontology development projects. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:1–16, 2012.

[158] E. P. B. Simperl and C. Tempich. Ontology engineering: a reality check. In *On the move to meaningful internet systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 836–854. Springer, 2006.

[159] A. Singhal. Introducing the knowledge graph: things, not strings. *Official Google Blog, May*, 2012.

[160] M. Sir, Z. Bradac, and P. Fiedler. Ontology versus database. *IFAC-PapersOnLine*, 48(4):220–225, 2015.

[161] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the WWW*, 5(2):51 – 53, 2007.

[162] K.-L. Skillen, L. Chen, C. D. Nugent, M. P. Donnelly, W. Burns, and I. Solheim. Ontological user modelling and semantic rule-based reasoning for personalisation of help-on-demand services in pervasive environments. *Future Generation Computer Systems*, 34:97–109, 2014.

[163] M. E. Smoot, K. Ono, J. Ruscheinski, P.-L. Wang, and T. Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3):431–432, 2011.

[164] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks. Optiquevqs: towards an ontology-based visual query system for big data. In *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*, pages 119–126. ACM, 2013.

[165] B. Srivastava and S. Kambhampati. The case for automated planning in autonomic computing. In *ICAC*, volume 5, pages 331–332, 2005.

[166] R. Stevens and P. Lord. Reification of properties in an ontology, 2010.

[167] M.-A. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. Noy. Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in protégé. In *Workshop on Interactive Tools for Knowledge Capture*, page 93. Citeseer, 2001.

[168] M. C. Suárez-Figueroa, A. Gomez-Perez, and M. Fernandez-Lopez. The neon methodology for ontology engineering. In *Ontology engineering in a networked world*, pages 9–34. Springer, 2012.

[169] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[170] V. Tamma, I. Blacoe, B. Lithgow-Smith, and M. Wooldridge. Introducing autonomic behaviour in semantic web agents. In *International Semantic Web Conference*, pages 653–667. Springer, 2005.

[171] E. Thomas, J. Pan, and Y. Ren. Trowl: Tractable owl 2 reasoning infrastructure. *The Semantic Web: Research and Applications*, pages 431–435, 2010.

[172] G. Thomson, G. Stevenson, S. Terzis, and P. Nixon. A self-managing infrastructure for ad-hoc situation determination. In *Smart Homes and Health Telematics*, 2006.

[173] A. Tolk and C. Blais. Taxonomies, ontologies, battle management languages-recommendations for the coalition bml study group. 2005.

[174] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *The Semantic Web-ISWC 2003*, pages 419–437. Springer, 2003.

[175] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *Automated reasoning*, pages 292–297. Springer, 2006.

[176] K. Twidle, N. Dulay, E. Lupu, and M. Sloman. Ponder2: A policy system for autonomous pervasive environments. In *Autonomic and Autonomous Systems, 2009. ICAS'09. Fifth International Conference On*, pages 330–335. IEEE, 2009.

[177] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal. Webpie: A web-scale parallel inference engine using mapreduce. *Web Semantics: Science, Services and Agents on the World Wide Web*, 10:59–75, 2012.

[178] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, pages 93–96. IEEE, 2003.

[179] C. Van Damme, M. Hepp, and K. Siorpaes. Folksontology: An integrated approach for turning folksonomies into ontologies. In *Proceedings of the ESWC Workshop bridging the Gap Between Semantic Web and Web 2.0*, 2007.

[180] E. Vassev and M. Hinchey. Modeling the image-processing behavior of the nasa voyager mission with assl. In *Space Mission Challenges for Information Technology*, pages 246–253. IEEE, 2009.

[181] D. Vrandečić and Y. Sure. How to design better ontology metrics. In *The Semantic Web: Research and Applications*, pages 311–325. Springer, 2007.

[182] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Autonomic Computing*, pages 70–77. IEEE, 2004.

[183] T. Walter, F. S. Parreiras, and S. Staab. An ontology-based framework for domain-specific modeling. *Software and Systems Modeling*, pages 1–26, 2014.

[184] H. H. Wang, N. Gibbins, T. Payne, A. Patelli, and Y. Wang. A survey of semantic web services formalisms. *Concurrency and Computation: Practice and Experience*, 27(15):4053–4072, 2015.

[185] T. D. Wang and B. Parsia. *CropCircles: topology sensitive visualization of OWL class hierarchies.* Springer, 2006.

[186] R. H. Weber and R. Weber. *Internet of things*, volume 12. Springer, 2010.

[187] M. Wilkinson. Designing an 'adaptive' enterprise architecture. *BT Technology Journal*, 24(4):81–92, 2006.

[188] F. Wuhib, M. Dam, and R. Stadler. Decentralized detection of global threshold crossings using aggregation trees. *Computer Networks*, 52(9):1745–1761, 2008.

[189] F. Xhafa, C. Paniagua, L. Barolli, and S. Caballe. A parallel grid-based implementation for real-time processing of event log data of collaborative applications. *International Journal of Web and Grid Services*, 6(2):124–140, 2010.

[190] F. Xia, L. T. Yang, L. Wang, and A. Vinel. Internet of things. *International Journal of Communication Systems*, 25(9):1101, 2012.

[191] H.-C. Yang. Bridging the WWW to the semantic web by automated semantic tagging of web pages. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, pages 238–242. IEEE, 2005.

[192] H. Zhang, H. Nguyen, E. M. Graciá, P. A. T. Solano, D. Zhang, N. Crespi, and B. Guo. Scalable multimedia delivery with QoS management in pervasive computing environment. *Supercomputing*, 65(1):317–335, 2013.