



Contract: 027617

SPICE

Service Platform for Innovative Communication Environment

FP6 - Integrated Project (IP)

Priority T2 – Information Society Technologies

WP N°: 4

Deliverable N°: 4.1

Title: Ontology Definition of User Profiles, Knowledge
Information and Services

Due date of deliverable: [November 30th, 2006](#)

Actual submission date: [December 13th, 2006](#)

Start date of project: 01/01/2006

Duration: 30 Months

Project coordinator name: Christophe Cordier

Organisation name of lead contractor for this deliverable: [University of Kassel \(UniK\)](#)

Revision: [v1.2](#)

PROPRIETARY RIGHTS STATEMENT

THIS DOCUMENT CONTAINS INFORMATION, WHICH IS PROPRIETARY TO THE **SPICE** CONSORTIUM. NEITHER THIS DOCUMENT NOR THE INFORMATION CONTAINED HEREIN SHALL BE USED, DUPLICATED OR COMMUNICATED BY ANY MEANS TO ANY THIRD PARTY, IN WHOLE OR IN PARTS, EXCEPT WITH THE PRIOR WRITTEN CONSENT OF THE **SPICE** CONSORTIUM THIS RESTRICTION LEGEND SHALL NOT BE ALTERED OR OBLITERATED ON OR FROM THIS DOCUMENT

Executive summary

DESCRIPTION OF THE DELIVERABLE CONTENT AND PURPOSE (MAX 1 PAGE)

This deliverable provides an overview of the ontologies specified for the use and exchange of information that is related to SPICE WP4 enablers. For this purpose, this deliverable briefly introduces the WP4 architecture and enabler groups, subsequently elaborates on the enabler specific requirements towards defining ontologies and finally depicts the enabler specific approaches and example instances of the resulting ontology definitions. In doing so, also specific aspects of ontology modelling with regard to uncertainty of knowledge are covered. Altogether, this deliverable deals with the following items:

- Introduction to WP4 architecture and enabler groups, and elaboration on enabler related requirements towards the definition of ontologies
- Description of SPICE WP4 approaches towards the definition of ontologies and relation to the state-of-the-art
- Comparison between different approaches to modelling relationship attributes for the representation of uncertainty and Quality of Context (QoC)
- Definition and visualisation of a set of ontologies that are related to SPICE WP4 enablers, in particular definition of user profile ontology, recommendations and learning ontology, service context ontology, presence ontology and physical space ontology
- Demonstration of ontology usage by example instances
- Relation of WP4 specific ontologies to SPICE reference model and Distributed Communication Sphere (DCS)

The resulting ontology definitions will enable the inter-working between different WP4 enabler groups, and between WP4 enablers and various other SPICE platform components. In the end, this semantic-based inter-working will enable the provision of meaningful and personalized service access to the end-users.

BRIEF DESCRIPTION OF THE STATE OF THE ART AND THE INNOVATION BROUGHT

The following table summarises the innovation brought by the specification of WP4 related ontologies.

Ontology	State-of-the-art in the area	Innovation
Ontology Framework	- No practical solutions for modelling relationship attributes, in particular Quality of Context (QoC)	- Adoption of W3C design patterns - Compatibility with existing ontologies such as the SPICE Mobile Ontology - Practical solution for modelling relationship attributes with ontologies

		<ul style="list-style-type: none"> - Re-use of Semantic Web standards and tools
Physical space ontology	<ul style="list-style-type: none"> - Several location related ontologies exist that make use of specifying distances 	<ul style="list-style-type: none"> - We modelled the physical space in a generic manner, where we made use of properties depicting that certain space classes are contained in other space classes. The “contained” relation is a transitive relation. This makes specifying distances and coordinates redundant for making a proper model that enables tracing people and objects in a building.
Service context ontology	<ul style="list-style-type: none"> - Several ontologies exist that cover parts of service context 	<ul style="list-style-type: none"> - Comprehensive service context ontology that comprises complete spectrum of SPICE related service context aspects such as context representation, context source information, context elements and Quality of Context (QoC)
User profile ontology	<ul style="list-style-type: none"> - Several top-level user profile schemata and domain specific user profile schemata exist - Mostly no or restricted means for including conditions (contextual constraints) that enable the activation of sub-profiles in specific user situations 	<ul style="list-style-type: none"> - Separates common service-independent user profile structure that includes activation conditions (contextual constraints) for specific context-dependent sub-profiles from service specific specification of the actual user data - Enables specification of context-dependent user sub-profiles for personalised service delivery that takes into account specific user preferences in specific user situations
Recommendations and learning ontology	<ul style="list-style-type: none"> - Ontologies have been applied in recommender systems and learning systems for specifying relationships in the actual input data (e.g. a category tree) - Recommendations (and learnt user models) have not been explicitly specified in most cases 	<ul style="list-style-type: none"> - Separates contents, services, and algorithms from each other. Previously, recommender systems have been closely tied to specific services and their contents - Recommendations can now be used directly by other components on the basis of the ontology definition
Presence ontology	<ul style="list-style-type: none"> - presence standards defined in IETF and OMA standards, used in IMS - no standardized PIDF compliant presence ontology 	<ul style="list-style-type: none"> - semantic description of standardized presence data - enabler for mapping between ontology-based systems and IMS



DEVIATION FROM OBJECTIVES

3.1 Description of the deviation

No deviation from the project work plan has occurred.

3.2 Corrective actions

n/a

List of authors

Full Name	Company Information
Abdelkrim Hebbar	Alcatel-CIT
Anna V. Zhdanova	University of Surrey
Christian Räck	Fraunhofer FOKUS
Herma van Kranenburg	Telematica Instituut
Josip Zoric	Telenor
Laurent Walter Goix	Telecom Italia
Marco Marengo	Telecom Italia
Martin Strohbach	NEC Europe Ltd.
Michael Sutterer	University of Kassel
Niels Snoeck	Telematica Instituut
Olaf Droegehorn	University of Kassel
Stian Alapnes	Telenor
Ying Du	University of Surrey

List of Reviewers

Full Name	Company Information
Anna V. Zhdanova	University of Surrey
Michael Sutterer	University of Kassel
Herma van Kranenburg	Telematica Instituut
Christophe Cordier	France Telecom
François Carrez	Alcatel CIT
Matthieu Boussard	Alcatel CIT

Document Information

Document Name:	Ontology Definition of User Profiles, Knowledge Information and Services
Document ID:	D4.1
Revision:	v1.2
Revision Date:	11 Dec 2006
Author:	See authors list
Security:	PUBLIC (PU)

Approvals

	Name	Company	Date	Visa
Technical Coordinator	Christophe Cordier	France Telecom	24/11/2006	
WP leader	Herma van Kranenburg	TELIN	24/11/2006	
Technical Manager	François Carrez	ACIT	24/11/2006	
Quality Manager				
Editor	Michael Sutterer	UniK	27/11/2006	

Documents history

Revision	Date	Modification	Authors
v0.1	01.08.2006	Initial table of content	Michael Sutterer
v0.2	19.09.2006	Update of document structure	Michael Sutterer
v0.3	29.09.2006	Update of document structure	Michael Sutterer
v0.40	24.10.2006	First integrated version including UniK review	Abdelkrim Hebbar, Anna V. Zhdanova, Christian Räck, Josip Zoric, Herma van Kranenburg, Marco Marengo, Martin Strohbach, Michael Sutterer, Niels Snoeck, Olaf Drögehorn, Stian Alapnes, Ying Du
v0.41	30.10.2006	Review of v0.40	Anna V. Zhdanova
v0.50	08.11.2006	Integrated version containing input from Fri 03, Mon 06 and Tue 07 Nov 2006 including UniK review	Abdelkrim Hebbar, Anna V. Zhdanova, Christian Räck, Josip Zoric, Herma van Kranenburg, Marco Marengo, Martin Strohbach, Michael Sutterer, Niels Snoeck, Olaf Drögehorn, Stian Alapnes, Ying Du
v0.51	14.11.2006	Review of v0.50	Anna V. Zhdanova
v0.60	15.11.2006	Input for section on Ontology Framework and Presence Ontology	Martin Strohbach, Niels Snoeck, Laurent Walter Goix
v0.61	16.11.2006	Integrated version containing input, updates, and reviews	Herma van Kranenburg, Marco Marengo, Abdelkrim Hebbar, Michael Sutterer, Ying Du
v0.62	20.11.2006	Additional input and clean-up	Martin Strohbach, Herma van Kranenburg, Christian Räck, Michael Sutterer, Stian Alapnes, Laurent Walter Goix



v1.0	20.11.2006	Updates and clean-up	Christian Räck, Josip Zoric, Michael Sutterer, Niels Snoeck
v1.1	29.11.2006	Few minor additions and changes	Anna V. Zhdanova, Michael Sutterer
v1.2	11.12.2006	Final release	Michael Sutterer

TABLE OF CONTENTS

1	INTRODUCTION	15
1.1	SCOPE OF THIS DOCUMENT.....	15
1.2	DOCUMENT STRUCTURE	16
2	SPICE REFERENCE MODEL.....	17
2.1	RELATION TO THE SPICE REFERENCE MODEL	17
2.2	RELATION TO THE DISTRIBUTED COMMUNICATION SPHERE.....	17
3	ARCHITECTURE OVERVIEW	19
3.1	KNOWLEDGE DISCOVERY AND EXCHANGE ENABLERS	22
3.1.1	<i>Description</i>	22
3.1.2	<i>Functionality</i>	24
3.2	PERSONAL INFORMATION ENABLER	24
3.2.1	<i>Description</i>	24
3.2.2	<i>Functionality</i>	24
3.3	KNOWLEDGE INTERPRETATION ENABLERS	25
3.3.1	<i>Description</i>	25
3.3.2	<i>Functionality</i>	26
3.4	ATTENTIVE SERVICES ENABLERS	27
3.4.1	<i>Description</i>	27
3.4.2	<i>Functionality</i>	28
3.5	INTER-WORKING ENABLERS.....	28
3.5.1	<i>Description</i>	29
3.5.2	<i>Functionality</i>	29
4	ONTOLOGY FRAMEWORK.....	30
4.1	REQUIREMENTS	30
4.1.1	<i>Modelling Relationship Attributes</i>	30
4.1.2	<i>Contradicting Knowledge</i>	32
4.1.3	<i>Compatibility</i>	32
4.1.4	<i>Tool Support</i>	32
4.2	STATE OF THE ART	33
4.2.1	<i>Probabilistic Extensions</i>	33
4.2.2	<i>Quality of Context</i>	33
4.2.3	<i>Practical Approaches</i>	33
4.2.4	<i>Tool Support</i>	35
4.3	APPROACHES	35
4.3.1	<i>Identifying Entities</i>	35
4.3.2	<i>Reification</i>	36
4.3.3	<i>RDF++</i>	42
4.3.4	<i>Comparison</i>	44
5	ONTOLOGY DEFINITIONS	46
5.1	PHYSICAL SPACE	46
5.1.1	<i>Visualisation of Physical Space Ontology</i>	46
5.1.2	<i>Physical Space Ontology Definition</i>	50
5.2	SERVICE CONTEXT.....	54
5.2.1	<i>Usage of Service Context Information in the SPICE Platform</i>	54
5.2.2	<i>Organisation of Service Context Information</i>	56
5.2.3	<i>Visualisation of Service Context Ontology</i>	57
5.2.4	<i>Service Context Ontology Definition</i>	60
5.3	USER PROFILES.....	63
5.3.1	<i>Requirements</i>	63
5.3.2	<i>State of the Art</i>	63

5.3.3	<i>Approach</i>	64
5.3.4	<i>Visualisation of User Profile Ontology</i>	66
5.3.5	<i>User Profile Ontology Definition</i>	70
5.4	RECOMMENDATIONS AND LEARNING	74
5.4.1	<i>Requirements</i>	74
5.4.2	<i>State of the Art</i>	74
5.4.3	<i>Approach</i>	76
5.4.4	<i>Visualisation of Recommendation and Learning Ontology</i>	80
5.4.5	<i>Recommendation and Learning Ontology Definition</i>	82
5.5	PRESENCE	87
5.5.1	<i>Requirements</i>	87
5.5.2	<i>State of the Art</i>	87
5.5.3	<i>Approach</i>	89
5.5.4	<i>Visualisation of Presence Ontology</i>	89
5.5.5	<i>Presence Ontology Definition</i>	95
6	CONCLUSION	96
7	REFERENCES	97

List of figures and tables

Figure 1: Typical <i>Knowledge Sources</i> reference model with highlighted service enabler components.....	18
Figure 2: Mapping of the SPICE enabler groups to types of information (ontologies)	20
Figure 3: <i>Knowledge Management Framework</i> overview	23
Figure 4: Subsystem of the KAPS	26
Figure 5: Architecture of the RDF-PIDF Gateway.....	29
Figure 6: Representing uncertain location information	31
Figure 7: Applying the W3C design pattern for n-ary relationships to our example	34
Figure 8: Example model using the reification approach	36
Figure 9: Entity hierarchy.....	37
Figure 10: Entity concept.....	37
Figure 11: <i>KnowledgeParameter</i> concept	38
Figure 12: Example <i>KnowledgeParameter</i> individual	38
Figure 13: <i>KnowledgeParameter</i> hierarchy	39
Figure 14: Sub-tree of <i>SensorReading KnowledgeParameter</i>	39
Figure 15: <i>BluetoothScan KnowledgeParameter</i> subclass	41
Figure 16: <i>GoalLink KnowledgeParameter</i> subclass	42
Figure 17: General approach.....	43
Figure 18: Example A-Box representation in RDF++.....	43
Figure 19: Illustration of (<i>Physical</i>) <i>Space</i> class and its properties.....	47
Figure 20: Illustration of properties <i>isOnFloor</i> , <i>hasAboveFloor</i> and <i>isAboveFloor</i>	48
Figure 21: Illustration of instances of class <i>Floor</i>	48
Figure 22: Illustration of <i>Space</i> , <i>Object</i> and <i>PhysicalEntity</i> classes and their properties..	49
Figure 23: Usage of service context information: several alternatives can be implemented, e.g. via context monitoring service (blue), or advertising module (red) (<i>SKPN</i> components).....	55
Figure 24: Direct usage of the service context (as knowledge consumers).....	56
Figure 25: Option in the organisation of the service context information	57
Figure 26: Top-level class hierarchy for service context.....	57
Figure 27: <i>ContextRepresentation</i> class and subclasses	58
Figure 28: <i>CommonContextInfo</i> class and subclasses	58
Figure 29: <i>AbstractContextComponent</i> class and subclasses.....	59
Figure 30: <i>ServiceContextElement</i> class and subclasses	59

Figure 31: High level user profile structure 65

Figure 32: User profile structure including *Conditional Profile Subsets* 66

Figure 33: Visualisation of profile ontology (part 1)..... 67

Figure 34: Visualisation of profile ontology (part 2)..... 68

Figure 35: *Profile Subset* instance of user Bob..... 69

Figure 36: *Feedback* class..... 76

Figure 37: *LearntRuleSet* and *LearntRule* classes 78

Figure 38: *Item* and *Feature* classes 79

Figure 39: *Recommendation* and *RecommendedItem* classes 79

Figure 40: Recommender subsystem and related concepts visualisation 80

Figure 41: Learning ontology visualisation 81

Figure 42: Snapshot visualisation..... 81

Figure 43: Presentity and related concepts 90

Figure 44: Person and related concepts (part 1) 91

Figure 45: Person and related concepts (part 2) 92

Figure 46: Service and related concepts 94

Figure 47: Device and related concepts 94

Figure 48: Place and related concepts 95

Table 1: Ontology usage in enablers and components..... 22

Table 2: Comparison between reification approach and RDF++ 45

Table 3: Log of weather conditions and golf-playing behaviour (example) 77

Glossary

Acronym	Signification
3GPP	3rd Generation Partnership Project
A4C	Authentication, Authorisation, Accounting, Auditing and Charging
ANN	Artificial Neural Networks
B2B	Business to Business
B2C	Business to Consumer
BN	Bayesian Networks
DCS	Distributed Communication Sphere
FOAF	Friend of a Friend
HTTP	Hypertext Transfer Protocol
IETF	The Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IMS-GW	IP Multimedia Subsystem-Gateway
KAPS	Knowledge Acquisition and Provisioning System
KB	Knowledge Broker
KMF	Knowledge Management Framework
KS	Knowledge Source
NTP	Network Time Protocol
OMA	Open Mobile Alliance
OTA	Open Travel Alliance
OWL	Web Ontology Language
PIDF	Presence Information Data Format
PM	Profile Management
QoC	Quality of Context
QoI	Quality of Information
QoS	Quality of Service
RDF	Resource Description Framework
RFC	Request for Comments
SIP	Session Initiation Protocol
SKPN	Service and Knowledge Push and Notification
SPICE	Service Platform for Innovative Communication Environment
SWBPD	W3C Semantic Web Best Practices and Deployment Working Group
UAProf	User Agent Profile
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WP	Work Package



Contract: IST-027617
Deliverable Report
Period covered: M00 – M11



XDM

XML Document Management

1 Introduction

The SPICE WP4 researches in and develops intelligent service platform solutions for user profile and context information management and anticipatory middleware functionality in the domain of context aware service platforms. These “intelligent” solutions are decomposed into different enabler groups. On the one hand these enablers provide relevant knowledge and contextual information concerning the end-user and his situation that is needed to tailor mobile services and applications. On the other hand they incorporate this information in intelligent service support mechanisms. The intelligence is strongly based on information (accessing and processing) about the end-user and his personal situation, his terminal, and on application-specific service-information. This information is enriched with semantic descriptions that allow for interoperability on the semantic level. The WP4 solutions therefore enable applications to give their users meaningful and personalized access to services at the right moment and furthermore enable pro-active session adaptation to the (dynamic) personal context of the users.

In particular, these enabler groups are:

- Knowledge Discovery and Exchange Enablers: provided by the *Knowledge Management Framework (KMF)* which is a framework for communication between and discovery of knowledge resources.
- Personal Information Enabler: manages personal information including context-dependent preferences and is implemented by the *Profile Management (PM)*.
- Knowledge Interpretation Enablers: provided by the *Knowledge Acquisition and Provisioning System (KAPS)* which takes care of deriving entailed knowledge.
- Attentive Service Enablers: composed of the *Service and Knowledge Push and Notification (SKPN)*, and the Predictors, allowing for pro-active responsiveness to (upcoming) changes in the environment.
- Inter-Working Enablers: components which allow the inter-working between the above enablers and the IMS world.

In order to enable the inter-working between these WP4 enabler groups, and also between WP4 enablers and various other SPICE platform components, all the information the enablers are exchanging has to be semantically described by ontologies. In the end, this semantic-based inter-working will enable the provision of meaningful and personalized service access to the end-users.

1.1 Scope of this document

The main focus of this deliverable is on the ontology that semantically describes the knowledge provided by WP4 *Knowledge Sources*. A *Knowledge Source* is any functional element that is able to provide knowledge. Thereto it obtains knowledge internally or from external *Knowledge Source(s)*, processes the obtained knowledge, and outputs its result as knowledge. Each *Knowledge Source* that becomes active needs to register itself at a *Knowledge Broker*. In doing so the *Knowledge Source* uses an OWL/RDF document that describes its capabilities in terms of the ontology. This (knowledge related) ontology is

used in order to convey semantics during knowledge exchange. For the exchange of knowledge, that is done as instances of ontology concepts, we use RDF statements.

Likewise when a knowledge consumer requires a certain type of knowledge, it specifies its needs in an OWL/RDF document that is linked to the ontology. Both operations where the reference to the ontology is made are specified in the interface of the *Knowledge Broker*.

1.2 Document structure

Section 2 of this document provides a summary of the SPICE Reference Model and describes its relation to the WP4 related ontology definitions. Furthermore it also describes its relation to the Distributed Communication Sphere (DCS) of WP3. Section 3 provides an overview of the WP4 architecture and related enabler groups, and shows a mapping between the enabler groups and the used information (ontologies). In section 4, general aspects with regard to the modelling of relationship attributes for the representation of Quality of Context (QoC) and uncertainty are discussed that affect all WP4 enablers. Subsequently, section 5 depicts several mostly enabler-specific ontology definitions, in particular the definition of a physical space ontology, service context ontology, recommendation and learning ontology, user profile ontology, and presence ontology. The specific ontology subsections discuss requirements and state-of-the-art in the related fields and subsequently describe the WP4 approach in showing example instances and visualisations of the ontology definitions.

2 SPICE Reference Model

In this chapter, we explain how the ontology work reported here relates to the general SPICE reference model, and discuss relations of this work to ontology work deriving from other mobile communication sub-domains and other project work packages.

2.1 Relation to the SPICE Reference Model

The ontology-related work brings to implementation the reference model as specified at the general project level [SPICE D1.3].

Specifically, the ontology work presented here focuses on the concepts and components as indicated in Figure 1. Figure 1 is a part of the SPICE general reference model, where knowledge-based service enabler components are placed in rectangles.

2.2 Relation to the Distributed Communication Sphere

An ontology specifying the *Distributed Communication Sphere* (DCS) [SPICE D3.1] has been delivered earlier in SPICE. DCS vocabulary is a machine readable schema intended for sharing knowledge and exchanging information both across people and across services/applications. DCS vocabulary covers domains related to mobile communications: persons, terminals, services, networks. The DCS Vocabulary definitions presented are written using RDF/OWL that makes it easy for software (both Web-based and mobile-oriented) to process basic facts about the terms in the DCS vocabulary, and consequently about the things described in DCS documents.

DCS vocabulary is intended to be a part of the Mobile ontology, intended to use within the SPICE project and beyond.

The Mobile ontology is coordinated as a joint effort between SPICE project participants that aims at construction of a high level ontology capturing generally applicable concepts of mobile communications domain.

Currently, Mobile ontology consists of three sub-ontology types:

- **Mobile ontology Core:** construction of a general purpose ontology covering various aspects of the domain (e.g. DCS in WP3),
- **Mobile ontology Standards:** adoption of existing standards (e.g. PIDF) to ontology formats,
- **Mobile ontology Support:** construction of ontologies that support or are relevant to mobile communications development (e.g. Quality of Context).

As well as the DCS vocabulary, the ontology work presented in this deliverable covers certain aspects in Mobile Communications domain, i.e. directly contributes to the Mobile ontology core. Specifically, the work focuses on delivering data models for knowledge-based support of context-aware mobile services.

Thus, the ontology work reported here generates contribution to the Mobile ontology with:

- Ontological specifications for data exchange between enablers developed in WP4 (e.g., for recommender, learner, profile manager) – contribution to Mobile ontology Core

- Ontological schema representing typical formats for presence (such as [RFC3863], [RFC4479], [RFC4480]) – contribution to Mobile ontology Standards
- Ontologies to characterise knowledge, such as quality of context specification – contribution to Mobile ontology Support.

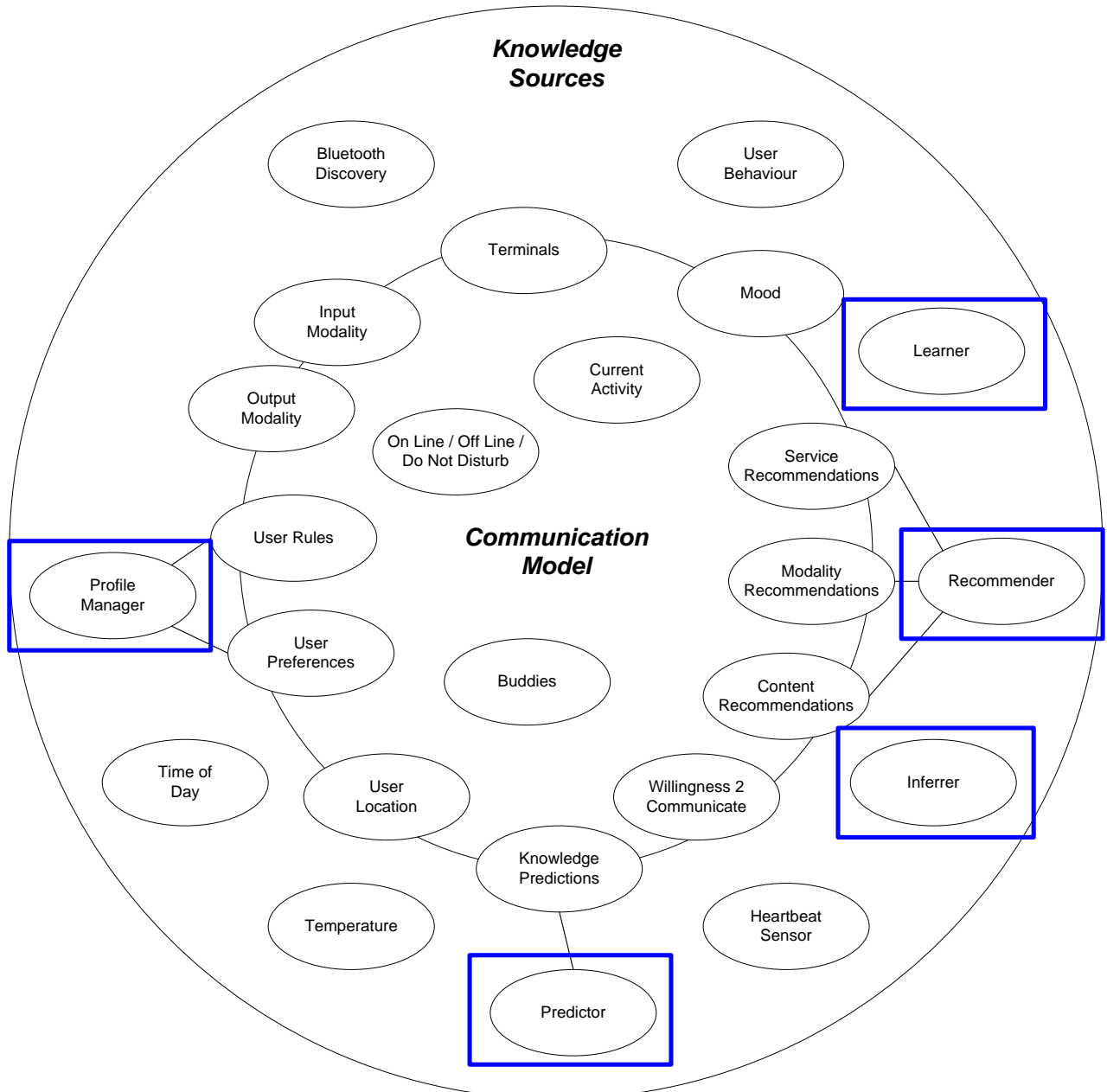


Figure 1: Typical Knowledge Sources reference model with highlighted service enabler components

3 Architecture Overview

The set of services enablers, which has been identified by WP4, consists of multiple enablers that are built on top of a common framework. This common framework is called the “*Knowledge Management Framework*” (*KMF*) and provides all necessary means for providing knowledge to interested parties, as well as for the exchange and discovery of knowledge within the SPICE platform.

The *KMF* is the basis of the so-called knowledge layer within the overall SPICE architecture. In WP4 we investigate key functions and mechanisms needed in adaptive, mobile middleware that support ubiquitous, attentive and context-aware distributed computing. Our intelligent enablers retrieve and process information from heterogeneous context, user profile, and service profile sources. The information is processed with advanced reasoning methods targeting at plausible and usable results. Scalable access mechanisms are addressed as well as the exchange of information between platforms and domains. Prediction techniques, mechanisms anticipating (foreseeable) changes, and integration mechanisms are studied yielding pro-active service enablers and extending the services with intelligence. These anticipatory service enablers allow for alertness and responsiveness to changes in the user’s environment, pro-actively triggering mobile services in advance to changes having actually occurred. These approaches are grouped as following:

- **Personal Information Management:** focuses on service platform provisioning of relevant and meaningful information and services based on user profile management and matching and profiling techniques.
- **Distributed Context Information Interpretation:** incorporates advanced inference engines that are able to use dynamic context information, user profiles, and application-specific knowledge in order to infer relevant knowledge for service providers, applications or end-users.
- **Attentiveness:** provides mechanisms that notify services of events (such as location or network Quality of Service (QoS) changes) that are relevant for them, and that can be specified by the service itself.

While this description focuses on the enabler approach, as followed in this document, it is important to understand that each enabler is provided by a set of components that encapsulates core functionality behind a simple interface. These components are explained in detail in [SPICE D4.4].

The five enabler groups are:

- Knowledge Discovery and Exchange Enablers: provided by the *Knowledge Management Framework (KMF)* which is a framework for communication between and discovery of knowledge resources.
- Personal Information Enabler: manages personal information including context-dependent preferences and is implemented by the *Profile Management (PM)*.
- Knowledge Interpretation Enablers: provided by the *Knowledge Acquisition and Provisioning System (KAPS)* which takes care of deriving entailed knowledge.

- **Attentive Service Enablers:** composed of the *Service and Knowledge Push and Notification (SKPN)*, and the Predictors, allowing for pro-active responsiveness to (upcoming) changes in the environment.
- **Inter-Working Enablers:** components which allow the inter-working with the IMS world.

The mapping between these enabler groups and the information (ontologies) they are dealing with is shown in Figure 2. On the left hand side, one can see the enabler groups, on the right hand side the related types of information (ontologies). For each type of information in the right hand side of this figure, except *Knowledge* information, section 5 provides a subsection where the related ontology is introduced in detail. The introduction of a *Knowledge* ontology is not needed, since *Knowledge* can be referred to as the composition of all the other information types. We will define a Recommendation and Learning Ontology for the use in Knowledge Interpretation Enablers, a User Profile Ontology for the use in Personal Information Enablers, a Service Context Ontology for the user in Attentive Service Enablers and a Presence Ontology for the use in Inter-Working Enablers. Furthermore, since all enabler groups deal with physical space (location) information, a common *Physical Space Ontology* will be defined.

It should be mentioned, that this list of ontologies is by no means complete. That is, WP4 may use further ontologies that will be developed or applied within the overall SPICE project. For instance, the *Attentive Service Enablers* may also deal with an ontology for the description of properties and capabilities of services such as OWL-S [OWL-S].

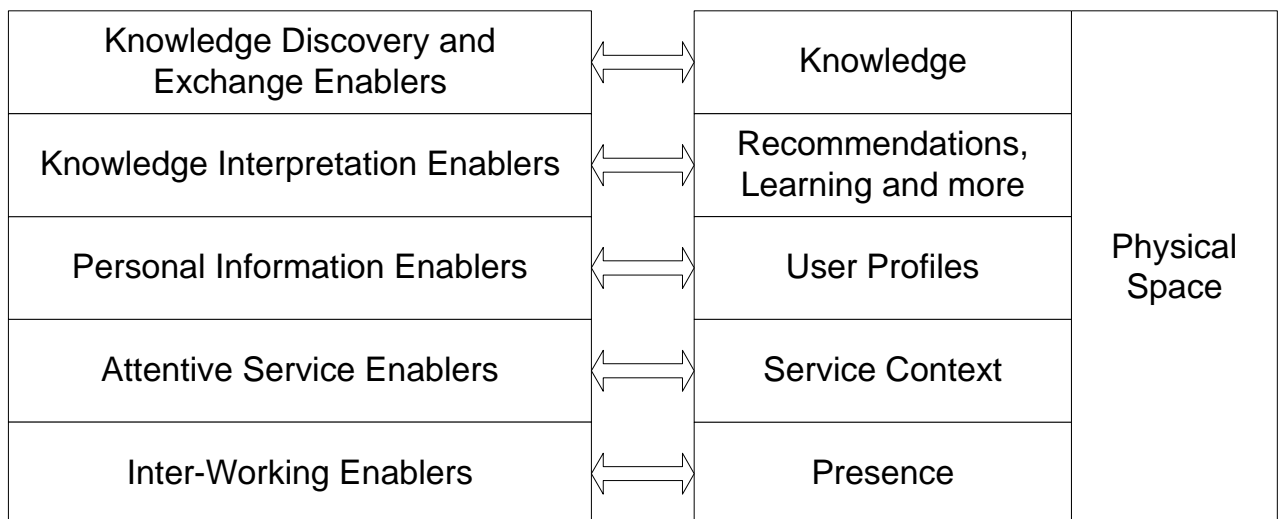


Figure 2: Mapping of the SPICE enabler groups to types of information (ontologies)

Table 1 depicts the ontology usage in WP4 enabler groups on an overview level. For each WP4 enabler group, the table includes the technologies used for RDF import / export, the used ontologies and schemata, and an explanation on the acquisition of used data. In addition, the last row of the table includes the same properties for *Ontology Matching Solution*. This solution is required in order to match concepts and properties of different

WP4 ontology parts. More details on the specific ontologies can be found in the appropriate subsections of section 5.

Name of enabler group or solution	RDF import	RDF export	Ontologies / schemata used	How do you get the data this component / enabler work with?
Knowledge Discovery and Exchange Enablers	Yes, with OWL API [OWL-API], Jena [Jena]	Yes, with Jena	RDF; Physical space ontology as part of SPICE Mobile ontology (using the knowledge parameter) as well as other parts of the SPICE Mobile ontology	Acquisition from <i>Knowledge Sources</i> logging context and delivering knowledge
Knowledge Interpretation Enablers	Yes	Yes	Recommender and learning ontology as part of the SPICE Mobile ontology as well as other parts of the Mobile SPICE ontology (e.g. location information)	Acquisition from different <i>Knowledge Sources</i> logging context and delivering knowledge (<i>learning through observation, reasoning</i>) as well as acquisition from all users typing into the system (<i>learning by feedback</i>).
Personal Information Enabler	Yes, with Jena	Yes, with Jena	User profile ontology as part of SPICE Mobile ontology as well as other parts of the Mobile SPICE ontology (e.g. physical space ontology)	Acquisition from all users typing into the system and from learned user behaviour

Attentive Service Enablers	Yes	Yes	Service context ontology as part of SPICE Mobile ontology as well as other parts of the Mobile SPICE ontology	Acquisition from <i>Knowledge Sources</i> logging context
Inter-Working Enablers	Yes	Yes	Presence Ontology as part of SPICE Mobile Ontology	Acquisition from <i>KMF</i> KS and IMS Presence Servers
Ontology Matching Solution	Yes, with OWL API, INRIA API [Euzenat2004], Jena	Yes, with Jena	Ontology alignment format, FOAF [FOAF], processes various domain ontologies incl. Mobile ontology	Acquisition from all users typing into the system, anything they bring in

Table 1: Ontology usage in enablers and components

The below subsections provide a short description and introduction to the enabler group functionalities as background knowledge to the subsequent definition of enabler-related ontologies. More details on the architecture and enabler groups can be found in [SPICE D4.4].

3.1 Knowledge Discovery and Exchange Enablers

Implemented by the *Knowledge Management Framework (KMF)*, see [SPICE D4.4], *this enabler* provides the communication framework on top of which the remaining enablers operate. This framework assists applications and other consumers of contextual information in handling and processing of contextual information, thereby shielding the underlying processes and optimal delivering of the requested contextual information to the consumers. Consumers of the *KMF* include applications, services and context handling components; in the remainder of this deliverable we will refer to these in short as “knowledge consumers”.

3.1.1 Description

A consumer gets the required contextual information from the *KMF*, without needing to know what kind of components are dealing with handling his request, nor with having to deal with separate parties that deliver subparts of the context, like a mobile operator or application provider. Hence consumers of the *KMF* can find and use the piece(s) of information that are relevant to them. Context in this sense refers to all available information that is relevant and that can be used to characterise the situation of an entity,

such as a human user. Our enablers enrich the context, and infer entailed information from it by processing lower level contextual and profile information. The enriched outcome is coined knowledge by us. Specialized functions, like reasoning with contextual information, inferring knowledge, handling personal preferences, and pushing tailored services to the consumers are supported and integrated by the *KMF*.

The *Knowledge Management Framework (KMF)* is a distributed middleware framework that facilitates the gathering and processing of contextual information for use by context-aware applications. In particular, the framework provides support for finding and transferring heterogeneous contextual information across multiple devices and administrative domains, and supplies distributed knowledge processing services. The *KMF* consists of an ad-hoc network of distributed *Knowledge Sources*, *Knowledge Sinks*, and *Knowledge Brokers*. Figure 3 illustrates the *KMF*, where solid lines show the flow of knowledge between different *Knowledge Sources* and applications, whereas dotted lines show the registration of the *Knowledge Sources* at a broker. A *Knowledge Source* is required to register a description of the information it can provide at the *Knowledge Broker*. When a knowledge consumer requests a particular type of information the *Knowledge Broker* replies with a reference to the *Knowledge Sources* that provide this information.

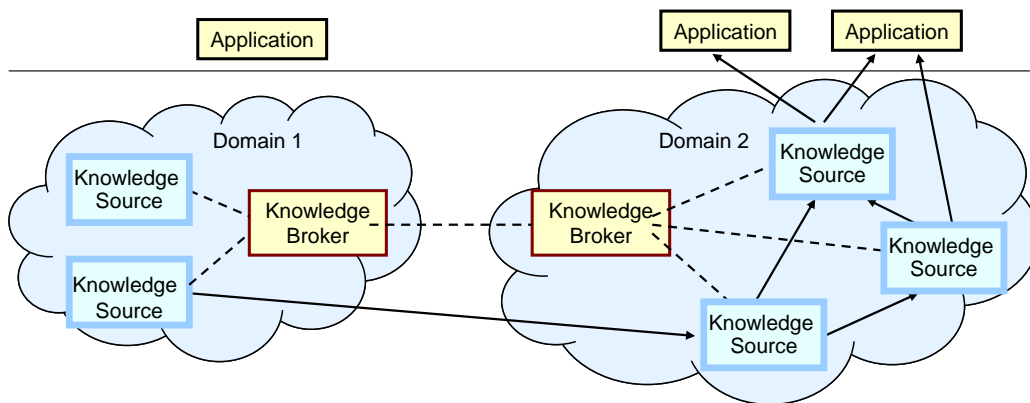


Figure 3: Knowledge Management Framework overview

The *KMF* supports the use of gathering and using sensor, profile and contextual information to derive entailed and predicted information, that we refer to as knowledge, and is provided by *Knowledge Sources*. This knowledge encompasses rules, facts, recommendations, preferences, predictions, and context data of various semantic levels. A common set of interfaces is used for semantic knowledge discovery and knowledge exchange. This enables (recurrent) re-use of various (interchangeable) context processing services that are provided by different providers, and internal make use of different reasoning techniques [vanKranenburg2005] [vanKranenburg2006]. The *Knowledge Sources* exchange the knowledge as instances of ontology concepts. Inter-working enablers take care of mapping different data formats that are supported by the *KMF*. This includes gateways that map enhanced P1DF/RDF and vice versa.

In a typical deployment scenario, *Knowledge Sources* are physically distributed over various devices, and a single device may host multiple *Knowledge Sources*, continuously feeding a wide range of contextual information types into the framework. In a multi-domain setting, *Knowledge Brokers* collaborate to locate *Knowledge Sources* in other domains. In the scope of the *KMF* a domain might represent an administrative domain, a device, etc.

3.1.2 Functionality

In summary the objectives of the *KMF* are to enable the SPICE platform to:

- Discover and have access to knowledge:
 - Where knowledge is entailed and predicted knowledge that is derived from lower level contextual information, sensor and personal information and encompasses rules, facts, recommendations, preferences, predictions, and context data of various semantic level
- Facilitate exchange of knowledge information:
 - Common knowledge format; knowledge is exchanged as instances of ontology concepts
 - Common set of interfaces allows for (re-current) re-use of various (interchangeable) context processing services that are provided by different providers, and internal make use of different reasoning techniques
 - (Different run-times, e.g. Java & .NET)
- Provide easy-to-use interfaces to applications and knowledge consumers
- Provide a plug-in architecture for new sources and interpreters of information, facilitating constellation of knowledge sources where knowledge from various sources is combined
- Shield underlying processes needed in getting the required knowledge, including token exchange and reasoning mechanisms, for knowledge consumers.

3.2 Personal Information Enabler

The SPICE service platform aims to provide means for personalising SPICE services. Therefore, it requires an enabler that manages user profiles and user preferences for various services. This functionality is provided by the SPICE *Profile Manager*.

3.2.1 Description

The *Profile Manager* enables the management of various user data. One of the most important functions of the *Profile Manager* is the provision of user data for the contextual personalisation of services. Thereby, the *Profile Manager* takes into account two basic requirements. First, the user data provision considers service-specific user preferences for the service to be personalised. Second, in order to achieve the best possible user satisfaction, the provided user data depends on the user's current context.

3.2.2 Functionality

The main functions of the *Profile Manager* are:

- Management of context-dependent user data

The *Profile Manager* deals with both dynamic context information and persistent user data. One key aspect of the *Profile Manager* is the integration of context-dependent user preferences into the user profile structure, since the user's behaviour and actions depend on his context. For instance, imagine a businessman with a mobile phone. Typically, he has different preferences concerning mobile phone usage for different contexts. While being in a meeting, he does not want to be notified of incoming news from his news feed service. On the other hand, when being by himself in his office, he immediately wants to be notified of all incoming news. The situational context of a user could e.g. be described with the user's current location, mood, the time of day, or any combination of different context variables.

- Management of service-specific user data

The SPICE platform deals with different services. These all may have different requirements but may use partly similar or even the same user data. From a user perspective, his user data should be shared by various services to ensure a convenient single sign-on. Hence, the profile manager provides means to manage user data for different services and aims to support the re-use and sharing of user data.

- Providing service and context-dependent user data

SPICE services require means for adapting to the user's wishes. This is supported by the provision of context-dependent user data that takes into account the user's wishes in specific situations. For this purpose, the *Profile Manager* internally uses searching and matching mechanisms in order to select the best matching user data. These searching and matching mechanisms take into account the user's current context that is received from other WP4 *Knowledge Sources*.

3.3 Knowledge Interpretation Enablers

To further advance existing approaches, the SPICE platform has to offer a strong support for discovery, exchange, and interpretation of semantically enriched information. By supporting these features the SPICE platform enables future service developers to realize truly next generation services that smartly adapt to the needs and wishes of their end-users. This adaptation functionality is provided by the SPICE *Knowledge Interpretation Enabler*, commonly called *Knowledge Acquisition and Provisioning System (KAPS)*.

3.3.1 Description

The main objective of the *KAPS* is to use readily available so-called lower order knowledge, such as context data, as well as additional higher order knowledge, such as recommendations, to deduce further knowledge about a given situation. In short, the *KAPS* infers knowledge by applying various learning, reasoning, and recommendations techniques. Figure 4 shows the four subsystems and identifies specific functional components that will be implemented.

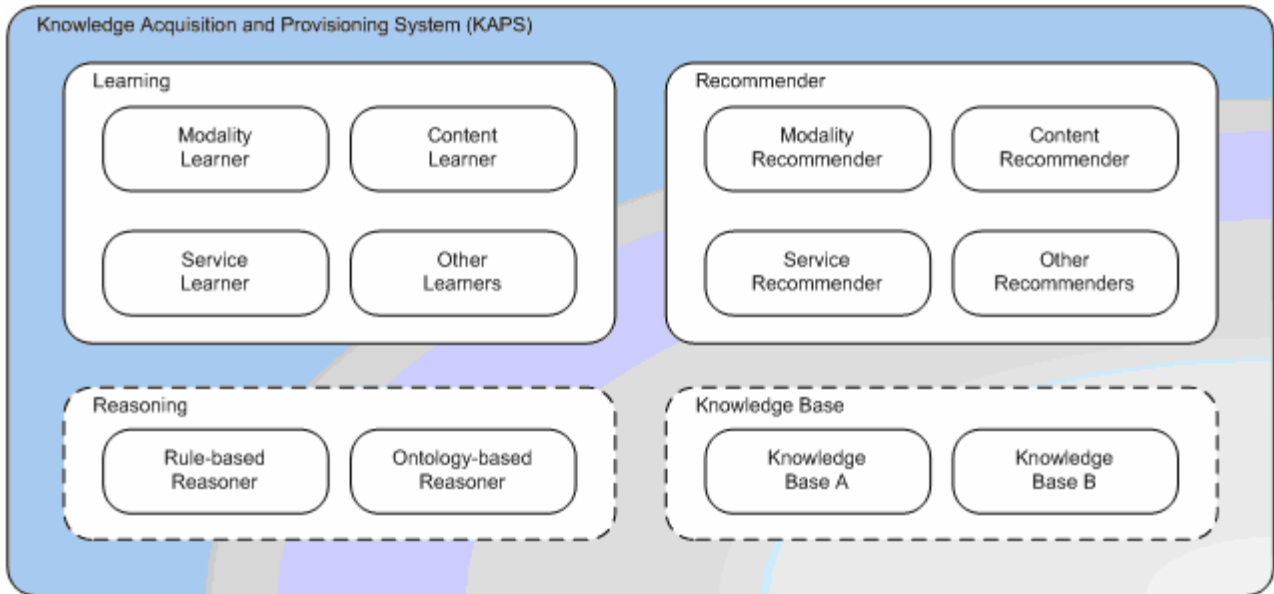


Figure 4: Subsystem of the KAPS

3.3.2 Functionality

The *KAPS* provides an open framework that enables intelligent service behaviour within the *SPICE* platform. The *KAPS* itself is part of the knowledge layer and is therefore built on top of the *Knowledge Management Framework (KMF)*. The knowledge layer is created through the collaboration of all components that are compliant to the *KMF* specification. To ensure the overall interoperability all components have to implement the *IKnowledgeSource* interface. This interface provides generalized means for the exchange of knowledge within the *SPICE* platform. The *KAPS* components are specialised *Knowledge Sources* from the *KMF*'s point of view. They fall into two main categories

- knowledge repositories, such as knowledge bases, and
- knowledge reasoners, such as learners, recommenders, and reasoners

The main functions of the *KAPS* are to enable the *SPICE* platform to

- acquire knowledge about its users, their interests, and their behaviour (*learning subsystem*),
- provide recommendations on modalities, contents, services, and actions (*recommender subsystem*),
- take actions that depend on what the system believes about the world, as opposed to just what the system explicitly knows about the world (*reasoning subsystem*).

The *learning subsystem* consists of all “learners” that have been deployed in the *KAPS*. These components acquire (learn) knowledge that is then available to other learners, recommenders, reasoners, and services. The actual acquisition (learning) process works either on the basis of provided feedback or through observation. Feedback is directly provided by services, whereas observations are gathered through a series of discrete

snapshots. These snapshots contain a set of variables that describe the data required for the learning process. The actual values of these variables are gathered from a set of (predefined) *Knowledge Sources*. The acquired knowledge is stored in a knowledge base.

The *recommender subsystem* consists of all “recommenders” that have been deployed in the *KAPS*. These components provide recommendations that consist of items that are relevant to either real-world entities or platform entities in given situations. Depending on the type of the actual recommendation, an item describes a particular modality, content, service, or action. All recommendations are provided on the basis of previously learnt knowledge as well as supplementary knowledge, such as context information or information about the items to recommend, if needed. The previously learnt knowledge is retrieved from a knowledge base, whereas the supplementary knowledge is gathered from other *Knowledge Sources*.

The *reasoning subsystem* consists of all “reasoners” that have been deployed in the *KAPS*. These components infer uncertain knowledge (believes) about the world, as opposed to just what the system explicitly knows about the world.

3.4 Attentive Services Enablers

The *Attentive Service Enablers* encompass service and knowledge push, and a set of predictors, which allow for intelligent proactive service provisioning and utilization.

3.4.1 Description

On the one hand, the *Service and Knowledge Push and Notification System (SKPN)* provides a solution that enables the notification of services and knowledge to a subscribed entity (e.g. end-user or external application) within the SPICE platform. The *SKPN* relies upon the *KMF* for the acquisition of knowledge and on the recommendation and learning mechanisms of the *KAPS*.

Users of *SKPN* mechanisms can be both end-user services and various service platform components and mechanisms, subscribed to the changes in the service context and the service related knowledge.

Attentive services are the subgroup of services (in the case of business to customer (B2C), or end-user services) and enabling services (in the case of business to business services (B2B)) that react on the change of the service context, or the knowledge that has an impact on the services / information delivered to the user. An example might be a change of the service context that enables the better Quality of Service (QoS) or Quality of Information (QoI) in already initiated services. Attentive services do not significantly change the already specified service architecture.

On the other hand, the group of *Predictors* provides solutions for predicting future knowledge. *Predictors* are software components that take past and current knowledge as input and estimate what the knowledge will be in a future point in time. Input knowledge will typically be based on learnt knowledge, i.e. a predictor will need to retrieve knowledge from a Learner component. Examples of predicted knowledge relate to the context of a user, location or object as well as knowledge about future services a user might be

interested in. Especially knowledge about future services can be used by a Recommender component and to implement attentive services.

3.4.2 Functionality

- The *SKPN* provides a solution that enables the notification of services and knowledge to a subscribed entity within the SPICE platform. In the case of attentive services, the subscribed entity might be:
 - a Service enabler (e.g. QoS manager, A4C manager, network enablers, service platform security enablers and similar)
 - a Service platform mechanism (service discovery, service composition, service broker)
 - an End-user service or even a 3rd party end-user service
 - any service architecture component that might want to subscribe to the context/information and knowledge changes

There is no significant difference in the way the *SKPN* serves the knowledge push and the notifications, provided that the exposed interfaces cover the needed parameters and methods.

The two components of the *SKPN* are the *Advertising Module*, and the *Context Monitor*. The specifics of knowledge push strongly inter-twin with the asynchronous subscribe knowledge method as specified in the *KMF*.

- *Predictors* provide solutions for the inference of future knowledge that is associated with uncertainty information. As input *Predictors* use past and current knowledge, in particular learned knowledge. The predictor subsystem consists of all *Predictors* that have been deployed in the *KAPS*.

3.5 Inter-Working Enablers

Inter-Working Enablers provide means for interacting with the IMS world. In SPICE, two kinds of approaches have been retained to manipulate the data concerning user presence. These data are exchanged between SPICE enablers either using:

- RDF representation
- PIDF representation

To allow the circulation of the user presence data between two components of different type, the SPICE *IMS-Gateway* – also called *IMS-Bridge* - has to provide a mean to make these data available to all components dealing with these two different formats.

So, the SPICE *IMS-Gateway* component can be seen as a component that abstracts the IMS Presence server for the other RDF SPICE enablers. It allows the coexistence of the two technologies mentioned above by ensuring the transmission of the user presence information between SPICE RDF and PIDF sides.

3.5.1 Description

Mainly, the *IMS-GW (IMS-Gateway)* behaves as a standard *KMF Knowledge Source* (specialized in knowledge storage) with respect to the RDF SPICE components and as a SIP-PIDF Presence Watcher component when communicating with the IMS Presence Server. On the IMS side, it allows to modify the Presence information, get this information or subscribe to the Presence Server to be notified each time it changes.

Once a request reaches the *IMS-GW*, the RDF data is translated to PIDF format thanks to the Data Format Mapping subsystem, see Figure 5. It should be mentioned, that it may not always be possible to translate RDF data to PIDF or vice versa. Hence, the *IMS-GW* also needs instructions in order to deal with such situations.

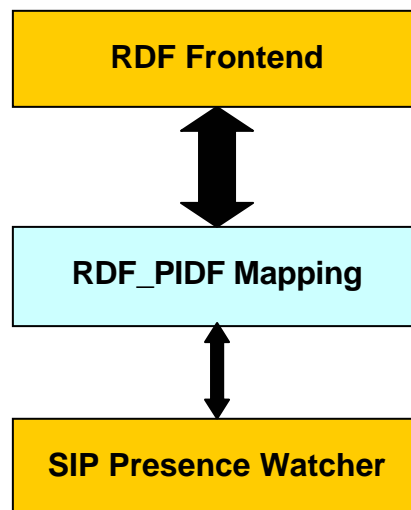


Figure 5: Architecture of the RDF-PIDF Gateway

3.5.2 Functionality

The SPICE *IMS-Gateway* provides the following features:

- Publication of the user presence information from *KMF* to the IMS Presence Server (in PIDF format), by acting as Knowledge Consumer in the *KMF* world
- Retrieval of the IMS user presence information, as well as subscription to, and related notification of the IMS user presence information changes, by acting as *Knowledge Source* in the *KMF* world

4 Ontology Framework

In the SPICE knowledge layer, an ontology is used in order to have a common understanding of knowledge information that is exchanged between components that produce and consume knowledge. This ontology is used in two distinct ways. First, the ontology semantically describes types of knowledge that *Knowledge Sources (KS)* may produce. A knowledge type description is used in the discovery of *KSeS* by knowledge consumers via *Knowledge Brokers (KB)*. In this process, each *KS* provides a *KB* with a type description of the knowledge it produces, and knowledge consumers query the *KB* for sources of knowledge types they are interested in. Second, the ontology is used to define the syntax and semantics of each piece of knowledge that is exchanged between a *Knowledge Source* and knowledge consumer.

This section describes a generic ontology framework that is used during discovery and knowledge exchange in the *KMF*, and hence affects all WP4 enabler groups. Our focus is on generating a generic structure that can be utilized to model real-world knowledge, including its intrinsic uncertainty and vagueness. In particular, we focus on the challenge to represent relationship properties between concepts that are required to infer the Quality of Context (QoC). We provide two approaches and compare them against each other. The work described in this section will provide the principles on which the enabler-specific ontologies described in following sections are based upon.

4.1 Requirements

In this subsection we identify four key requirements for the knowledge ontology. First, it should be possible to associate attributes to relationships between concepts such as temporal or uncertainty information. This is required to infer the Quality of Context (QoC). Second, it should be possible to model contradicting knowledge as provided by independent *Knowledge Sources*. Third, it should be compatible with other ontologies, i.e. it should be possible to integrate with other ontologies such as the SPICE Mobile Ontology. Finally, it should be possible to use existing design and reasoning tools.

4.1.1 Modelling Relationship Attributes

Acquiring knowledge about the real world is an intrinsically uncertain process. This means that *Knowledge Sources* may not be able to provide definitive information about some phenomenon they are meant to convey. Consider for example a location system that can only provide positioning information with an accuracy of more than 10 meters. Such a system will in general not be able to detect which room a user is located in. In addition, other factors like the reasoning process itself may introduce additional uncertainty.

An important observation is that uncertainty is often a property of a relationship rather than a property of the concepts themselves. For example, this means that we should be able to represent the location of a person as depicted in Figure 6. Here, the confidence of Bob being in Room1 or Room2 respectively is expressed using a probability value p between 0 and 1. The two relationship instances could be a result of positioning information of two different location systems.

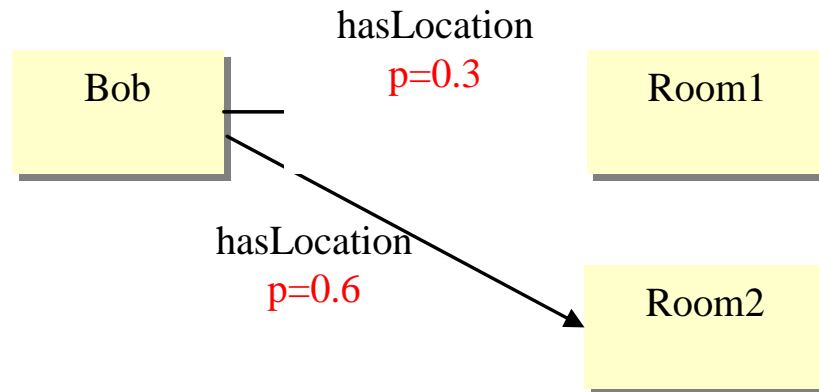


Figure 6: Representing uncertain location information

Note that the probability attribute p cannot be modelled as a property of the person Bob nor its location *Room1* or *Room2*. This is due to the fact that the probability provides additional information about the relationship rather than the linked concepts.

Within SPICE the modelling of relationship attributes will be of great importance as knowledge consumers may be able to interpret the knowledge more accurately. Especially if other attributes than uncertainty are used, the quality of the context can be inferred. The concrete attribute types depend on the characteristics of the knowledge at hand. We are currently considering the following attributes:

- **Probability** – A value on a fixed scale that defines the probability of correctness of a piece of knowledge
- **Confidence** - A value on a fixed scale that describes how confident the *Knowledge Source* is about a piece of knowledge
- **Timestamp** - A time value that marks the time a *Knowledge Sources* created this piece of knowledge
- **Temporal Validity** - A time interval that specifies when a piece of knowledge is valid, where the time interval may be in the past, current as well as future. For example a *Predictor* would provide knowledge that is valid in a future point in time.
- **Accuracy** - A value specified in the unit of the knowledge value that describes possible deviations from the knowledge value.

Further possible attributes could include the identification of the *Knowledge Sources* that provide this context and the derivation history, which lists the *Knowledge Sources* that were required to provide this context. Information like timestamps, the knowledge source and derivation history is important information for inferring the quality of a context: older context could be considered of lesser quality than newer context; identifying the knowledge sources can help to infer its trustworthiness and context that involved a large number of knowledge sources may be considered less reliable. However, the specification of attributes should be made with care, since it may have impact on the *KMF* functionality. For instance, the identification of *Knowledge Sources* decreases overall

flexibility, since a *Knowledge Source* can no longer be replaced by another *Knowledge Source* that provides the same type of knowledge, without affecting knowledge consumers.

4.1.2 Contradicting Knowledge

The model must allow multiple, possibly (partly) superfluous or contradicting pieces of knowledge to co-exist simultaneously. The following example illustrates the principle of possibly (partly) superfluous and contradicting knowledge in the domain of location determination.

(KS₁) Bob hasLocation Room1 at 11:15h with Probability 0.3 and Accuracy 10m
(KS₂) Bob hasLocation Room1 at 11:15h with Probability 0.6 and Accuracy 30m
(KS₃) Bob hasLocation Room2 at 11:15h with Probability 0.6 and Accuracy 30m

Each of the above statements may have been produced independently by distinct *Knowledge Sources* (designated KS₁ to KS₃ in this example), where each *Knowledge Source* uses a different approach or input knowledge to estimate an entity's location. All the statements describe the location of Bob at a certain time. *Knowledge Sources* 1 and 2 agree on its location, but their knowledge differs in probability of correctness and accuracy. Thus their knowledge about the location of Bob may be combined to create a more precise position estimation. In contrast, the probability and accuracy of the knowledge produced by *Knowledge Sources* 2 and 3 matches, but they do not agree on Bob's location, and therefore their knowledge is contradicting. Without additional knowledge, a knowledge consumer is unable to determine which of the *Knowledge Sources* is correct, and therefore both types of statements should be able to co-exist in the knowledge model.

4.1.3 Compatibility

The knowledge layer ontology is primarily used to define the semantics of knowledge types and instances exchanged between *Knowledge Brokers*, *Knowledge Sources*, and knowledge consumers. The agreement upon such a shared ontology is critical for the internal operation of the *KMF*. Within *SPICE* the *KMF* together with *KAPS* is an important enabler. Any decisions about knowledge representation will directly influence other *SPICE* components. It is therefore a desirable property that the ontology defined in this work package is compatible with other ontologies that are concerned with the representation of knowledge within the *SPICE* platform. This applies in particular to the *SPICE Mobile Ontology*.

The optimal case would be to re-use and extend concepts of existing ontologies. But as a minimum requirement it must be possible to map between the concepts of the *KMF* ontology and external ontologies.

4.1.4 Tool Support

Ontology design decisions, such as language selection and model characteristics, should whenever possible take the capabilities of existing tools into account. Several tools for designing and reasoning with ontologies are readily available. Three types of tools are required for efficient practical use of ontologies. First, tools should be available for modelling the ontology at design time. Such tools are typically used to define valid

concepts, properties and restrictions, i.e. T-box (Terminology-box) definitions. Second, tools for generating and parsing ontology instances - A-box (Assertions-box) statements - at runtime should be available. Third, for *Knowledge Sources* that perform reasoning on knowledge in ontological form, tools for reasoning with ontology instances in the defined format should be available.

4.2 State of the Art

How quality of context and in particular uncertainty information can be modelled with ontologies is currently still an open question. OWL is a widely used ontology language used for semantic web applications with good tool support. The main problem is however that ontology languages like OWL are based on Description Logics, a logic that does not support the assignment of attributes to properties. In this subsection we provide a short overview about the state of the art of theoretical and practical work.

4.2.1 Probabilistic Extensions

Ding and Peng propose a probabilistic extension to OWL that models uncertainty of class memberships [Ding2004]. Using Bayesian Networks they are able to model conditional class membership probabilities. However uncertainty of other relationships is not supported.

PR-OWL is a more general probabilistic extension to OWL with which uncertainty of relationship between concepts can be expressed [daCosta2005]. A similar approach is proposed by Pool et al. who argue for extending OWL due its widespread use and tool support and the superficial simplicity to implement probabilistic extensions [Pool2005]. However, they conclude with the remark that using OWL to implement their probabilistic concepts resulted in a high representational complexity and they raise the question whether OWL provides the correct foundation for uncertainty reasoning.

Other probabilistic extensions include probabilistic RDF extensions [Fukushige2005] and the development of a fuzzy description logic for the semantic web [Straccia2005]. A web interface implementation based on fuzzyDL has been developed [FuzzyDL].

In summary research on probabilistic ontology extension is not yet in a stage that we could integrate the results in our ontology framework. In particular, results are largely theoretic and development tools are hardly available.

4.2.2 Quality of Context

A more general approach that goes beyond probabilistic extensions has been proposed by Zimmer [Zimmer2005]. Here the general idea is to associate context with genomes that include information such as the number of derivation steps. Zimmer focuses on the algorithm to infer the QoC. However, the information represented in the genome provides useful clues about what relationship attributes should be represented in the ontology.

4.2.3 Practical Approaches

In this subsection we present practical work that has inspired our approaches. The first refers to the N-ary design pattern for OWL proposed by the W3C Semantic Web Best Practices and Deployment Group (SWBPD). The second refers to a well known context-

aware system that deals with uncertainty information and uses ontologies to semantically describe the information exchanged.

OWL and N-Ary Design Patterns

The ontology language OWL provides a means to model knowledge in terms of entities (expressed in concepts) and relationships (called properties). Unfortunately, OWL is not expressive enough to specify relationship attributes as required by the SPICE platform (see Figure 6).

The W3C addresses this limitation of OWL with a design pattern for N-ary relationships that is meant to address this and similar issues [N-ary Relations]. Figure 7 shows how the design pattern can be applied to our example. A detailed explanation of this design pattern is explained in our reification approach that uses this pattern (see Section 4.3.2).

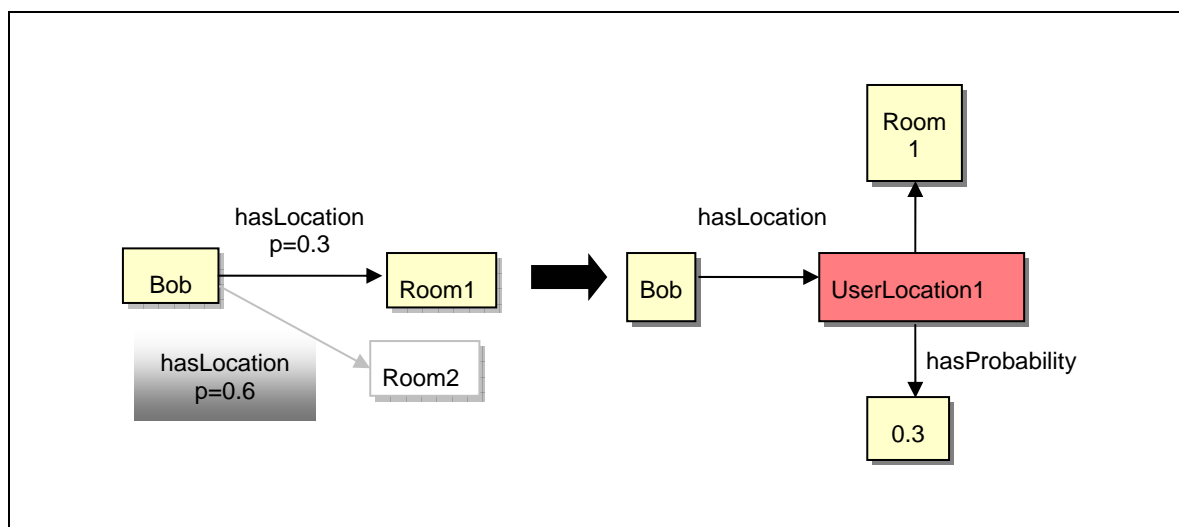


Figure 7: Applying the W3C design pattern for n-ary relationships to our example

Gaia

Gaia is a middleware infrastructure that manages resources in computer-augmented spaces such as homes, offices, cars, malls or airports to provide useful end-user services [Roman2002]. Central to Gaia is the representation of context information in subject-object or subject-verb-object format such as temperature(room 3231,is,98F). Context triples directly map to first order logic predicates based on which Prolog style reasoning can be performed. The Gaia middleware uses ontologies to describe the semantics of predicates. The ontology is defined in DAML+OIL, a predecessor of OWL. In Gaia, the ontology is used in three ways: (1) to check the validity of arguments, i.e. for checking the number of arguments a predicate can take, (2) to enable interoperation between different systems, i.e. by defining translation between different ontologies, and (3) to define how various system components interpret the same piece of context.

Uncertainty does not seem to be modelled with the ontology, but by attaching confidence values to actual instances of context information: $\text{prob}(\text{location}(\text{carol}, \text{in}, \text{room3231})) = 0.5$. Probabilistic logics and Bayesian networks are used to reason about uncertainty.

4.2.4 Tool Support

Unfortunately, most tools only support a subset of the theoretically defined languages and modelling and reasoning operations.

4.3 Approaches

In this section we present two alternative approaches for an ontology framework. The core challenge for both approaches is to find a pragmatic solution that addresses the lack of standardized ontology languages that allow the description of relationship attributes (see previous section). The first approach is based on the principle of reification (not to be confused with RDF reification described in the original RDF primer [RDF-Primer]), where additional information is added to a property by modeling the property as a concept. The second approach is based on adding relationship attributes to ontology instances represented in an extended version of RDF.

4.3.1 Identifying Entities

How is a user (or any other entity) identified in the knowledge layer ontology? The following possibilities exist:

- using some URI:

```
<user rdf:id="http://server.domain.co/...#someid">
```

- using properties:

```
<user>  
  <identifier>someone@somedomain</identifier>  
  <identifier>someone@someotherdomain</identifier>  
  ...  
</user>
```

The first method has the advantage that all properties assigned to a user with the specified id will be merged “automatically”, since all properties relate to the same URI with which the user is identified, it does however assume an agreed global naming scheme. Automatic merging can be a disadvantage in cases where it is specifically unwanted, since it cannot be ‘disabled’; it is a result of assigning a meaning to the URI of an element.

The second case has the advantage that multiple instances pertaining to the same user can be created, which is useful for e.g. properties that change over time or contain conflicting information. It also enables a user to have multiple identities.

Since there are cases where automatic merging is unwanted (like history tracking etc) we propose to use the second approach (using properties). This means that knowledge messages must not contain URIs for entities; they should be anonymous nodes with respect to RDF. It also means that any entity, which has to play a part in a knowledge message, must have an identifier property. The same user, device, location, etc. can be registered under multiple identities.

Examples of identifiers are:

- User: userID@domainID
 where userID can be a username, telephone-number, etc (like HvK@telin.nl and 0534850423@telin.nl for Herma)
- Device (with an access network interface): deviceID@domainID
 A typical DeviceID is its IP-address, MAC-address, BluetoothID, IMEI number,
- Service: serviceID@domainID
 A typical ServiceID is a ip/port combination or URL through which the service can be invoked
- Location: locationID@domainID
 This refers to *logical* locations (e.g. rooms, well-known places) (e.g. enschede@TELIN.nl, or demo.enschede@TELIN.nl)

With respect to the user profile management system: Each application or service (*Knowledge Source*) can have its own sub-profile including an identifier used for processing profile or context information of that person. This information can then only be requested by the service that created the sub-profile or other services (like aggregation services) that are given permission to read it.

4.3.2 Reification

In OWL, a direct relationship between concepts does not allow meta-data to be attached to the relationship, because OWL does not allow properties to be assigned to properties. Therefore, in this approach relations between entities are made indirect to be able to assign meta-data. In short, each direct relationship (property) is replaced by a concept that represents the relationship, and meta-data of the relationship is modeled as properties of this concept. For example, the location example of before would be modeled as illustrated in Figure 8, where the additional time and probability metadata of the location are modeled as properties of an intermediate concept. This technique is based on a design pattern proposed by the W3C community, called reification [N-ary Relations] (not to be confused with RDF reification described in the original RDF primer [RDF-Primer]).

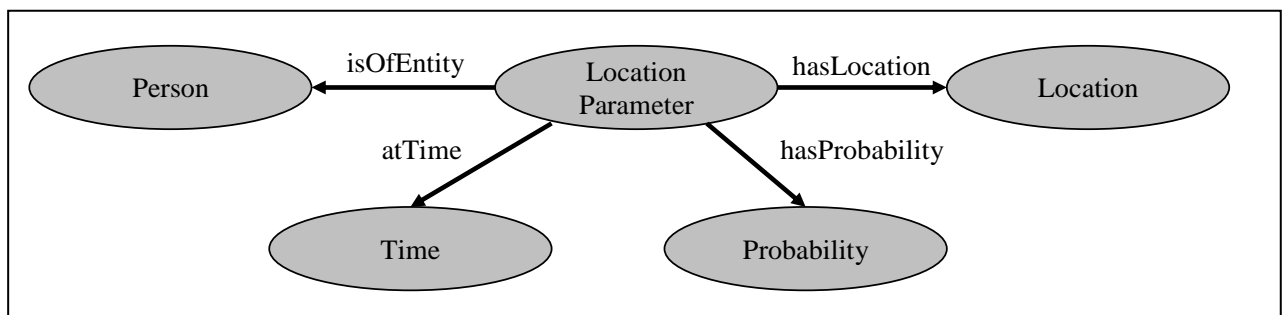


Figure 8: Example model using the reification approach

When using the reification approach, the ontology contains two types of concepts: one type represents concepts in the world the same way as in the standard modelling approach; the other type represents relationships and the accompanying meta-data. To distinguish between both types of concepts, two top-level concepts are introduced. The *Entity* concept is the root concept of all concepts that represent an entity in the real world. The *Entity* concept (see Figure 10) subsumes *Physical* and *NonPhysical* entities, which in turn subsume other concepts like activity, space, person, object, etc (see Figure 9).

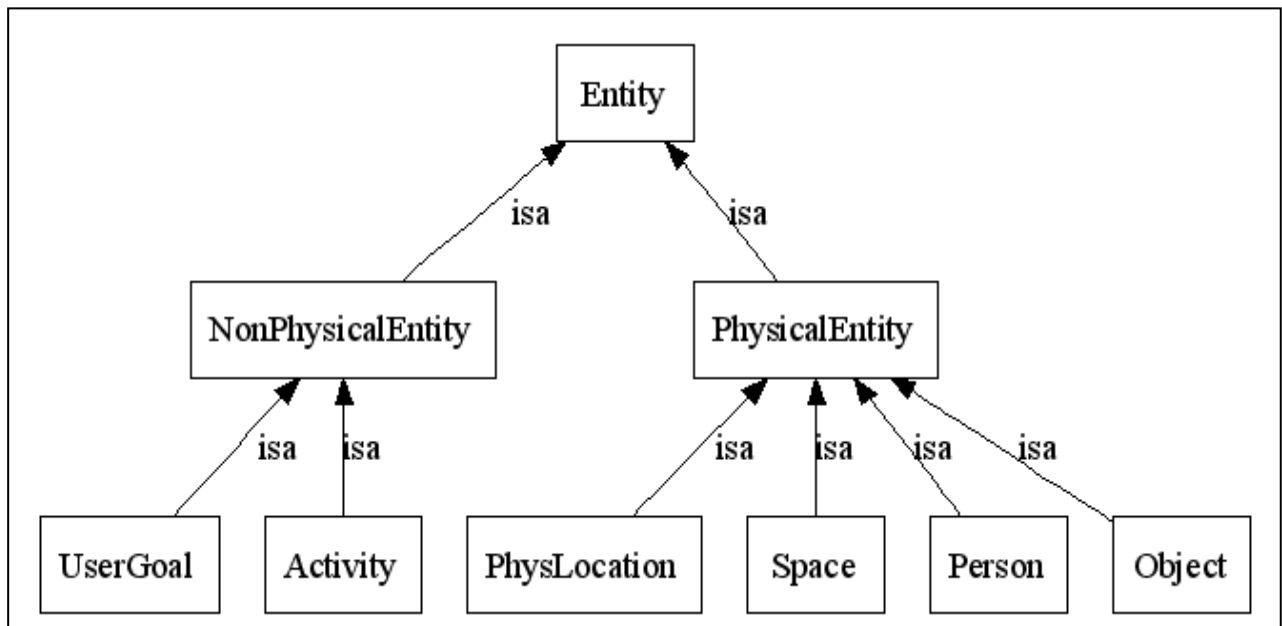


Figure 9: Entity hierarchy

Entity		
identifier	String*	
hasKnowledge	Instance*	KnowledgeParameter

Figure 10: Entity concept

The *KnowledgeParameter* concept is introduced as the root concept of all relationships. Specifically, the *KnowledgeParameter* concept represents the indirect links between concepts that are subsumed by the *Entity* concept; hence it shows which entities are related. The entities that participate in a relationship are linked to the *KnowledgeParameter* concept via the *isKnowledgeOf* property. Meta-data that is generic to all relationships is represented by additional properties of the *KnowledgeParameter* concept. These properties include *atTimestamp*, which indicates at which time the relationship holds, and *hasProbability*, indicating the probability of correctness of the relationship. The *KnowledgeParameter* concept and its properties are illustrated in Figure

11, and an example individual of a relationship between two concepts is shown in Figure 12.

KnowledgeParameter		
hasProbability	Float	
atTimestamp	String	
isKnowledgeOf	Instance*	Entity

Figure 11: KnowledgeParameter concept

```

<KnowledgeParameter>
  <atTimestamp>2006-05-04T13:39:20</atTimestamp>
  <hasProbability>0.7</hasProbability>
  <isKnowledgeOf>
    <entity>
      <identifier>...</identifier>
    </entity >
  </isKnowledgeOf>
  <isKnowledgeOf>
    <entity>
      <identifier>...</identifier>
    </entity >
  </isKnowledgeOf>
</KnowledgeParameter>

```

Figure 12: Example KnowledgeParameter individual

Although the *KnowledgeParameter* concept shows which entities are related, it does not convey the meaning of the relationship. To specify the meaning of a relationship between entities, the *KnowledgeParameter* concept is subclassed, where each subclass represents a distinct type of relationship that may hold between entities. The class hierarchy, which can be extended upon as more types of knowledge need to be modelled, is shown in Figure 13 and Figure 14. Advantages of the subclassing technique include the ability to add relation specific properties to a knowledge relation. By also subclassing the *isKnowledgeOf* property, the types and number of entities that can be involved in a particular relationship can be restricted as well. Example *KnowledgeParameter* and *isKnowledgeOf* subclasses are described in section 4.3.2.1.

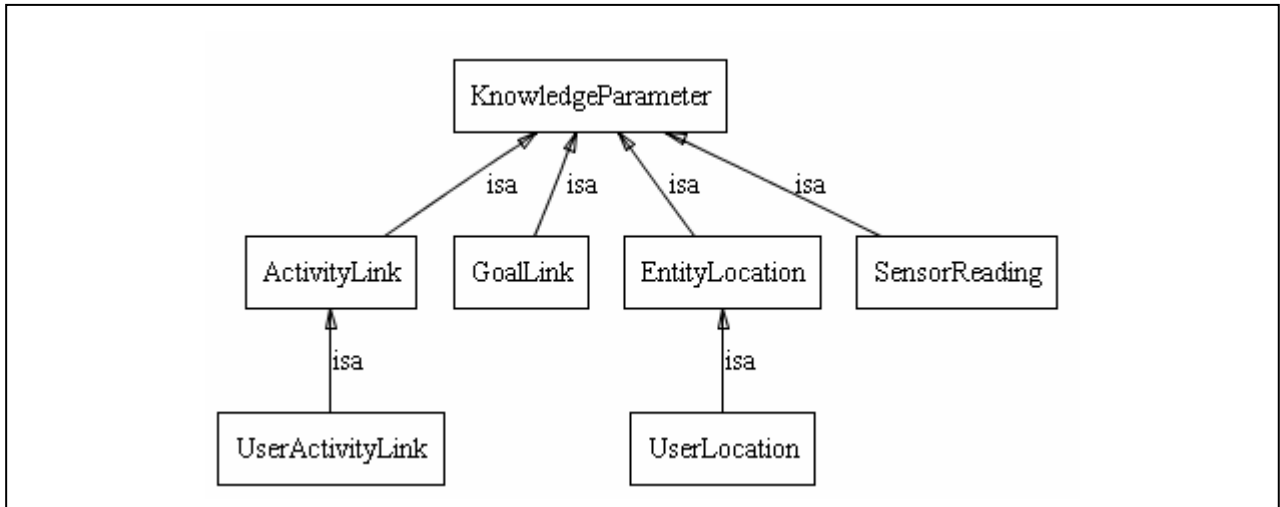


Figure 13: *KnowledgeParameter* hierarchy

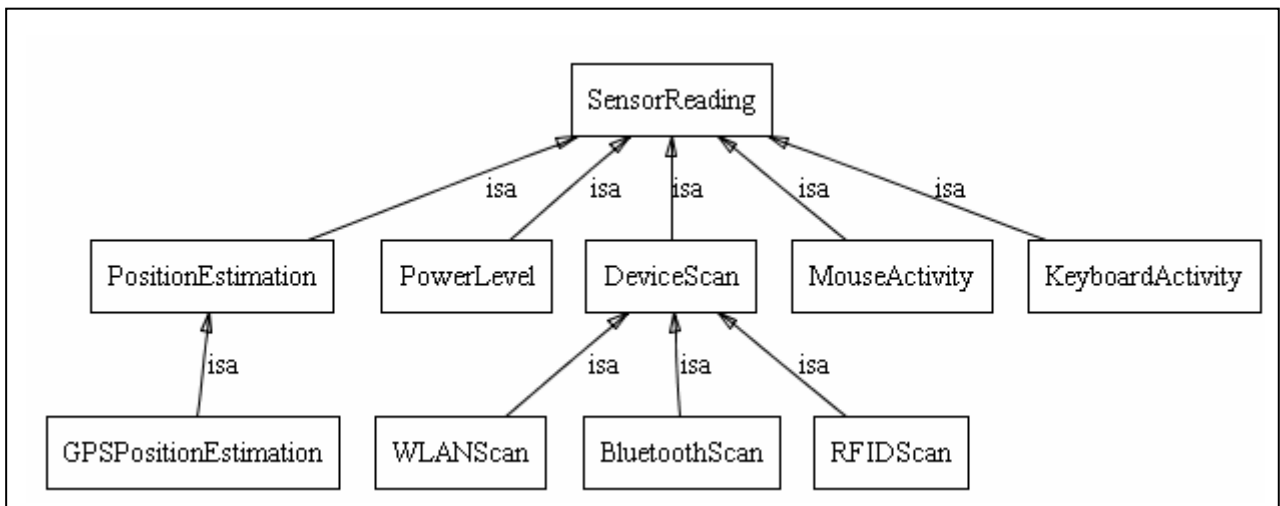


Figure 14: Sub-tree of *SensorReading KnowledgeParameter*

Currently open design decisions concerning the reification approach that will be investigated for the final SPICE version include:

- Whether a *KnowledgeParameter* individual may have a unique ID, which can be used for tracing purposes.
- Whether Quality of Context (QoC) meta-data should be modelled as direct properties of the *KnowledgeParameter*, or whether there should be an additional QoC concept that is related to the *KnowledgeParameter* concept via a single *hasQoC* property.
- Whether the time during which a context relationship holds, as modelled by the *atTimestamp* property, can be synchronized accurately enough between multiple *Knowledge Sources* to allow distributed temporal reasoning. One possible solution is to synchronize time via a centralized time server (NTP).

In the following text some examples will be given for the different types of knowledge that can be modelled with the described reification approaches. These examples do not cover the entire knowledge layer ontology, but are intended to be illustrative of the core principles of modelling knowledge.

4.3.2.1 Examples

User Location

A typical example of context information is the location of a person. In this particular example, a person has the role of a user of the SPICE platform, and therefore is represented by a *User* concept, which is a specialization of the *Person* concept. This example also illustrates the use of the *Room* concept, a subclass of the *PhysicalLocation*, which models a specific type of logical location.

An example *KnowledgeParameter* subclass instance that models the fact that a user is in a room at a particular time with a particular probability is shown here:

```
<UserLocation>
  <atTimestamp>2006-05-04T13:39:20</atTimestamp>
  <hasProbability>80</hasProbability>
  <isLocatedIn>
    <Room>
      <Identifier>B3.08B@telin.nl</Identifier>
    </Room>
  </isLocatedIn>
  <isLocationOf>
    <User>
      <Identifier>remco.poortinga@telin.nl</Identifier>
    </User>
  </isLocationOf>
</UserLocation>
```

Here, the *UserLocation* concept is a subclass of *KnowledgeParameter*. The *isLocatedIn* and *isLocationOf* properties are subsumed by the generic *isKnowledgeOf* property, where their ranges are restricted to *Location* and *User* concepts respectively.

An example SPARQL [SPARQL] query that retrieves this knowledge from an ontology filled with location information would look like this:

```
SELECT ?person ?room ?prob ?ts WHERE {
  ?p rdf:type spice:Person .
  ?p knowledge:identifier ?person .
  ?ul rdf:type knowledge:UserLocation .
  ?ul knowledge:isLocationOf ?p .
  ?ul knowledge:isLocatedIn ?r .
  ?r knowledge:identifier ?room .
  ?ul knowledge:probability ?prob .
  ?ul knowledge:timestamp ?ts . }
```

Bluetooth Scan

One of the ‘sensors’ that may be used by a mobile terminal in the SPICE platform is a Bluetooth radio. One of the capabilities of this technology is to scan for other (discoverable) Bluetooth radios, which provides knowledge on Bluetooth devices that are within roughly a 10 meter range of the terminal. This knowledge of nearby Bluetooth devices is represented by another subclass of the *KnowledgeParameter* concept: *BluetoothScan*.

A *BluetoothScan* is a specific type of *DeviceScan*; which in itself is a subclass of a *SensorReading* (refer to Figure 13). A *BluetoothScan* individual specifies the relationship between a device equipped with Bluetooth (*DeviceWithBT* in Figure 15) that performed the scan as well as the devices that were discovered by that Bluetooth device.

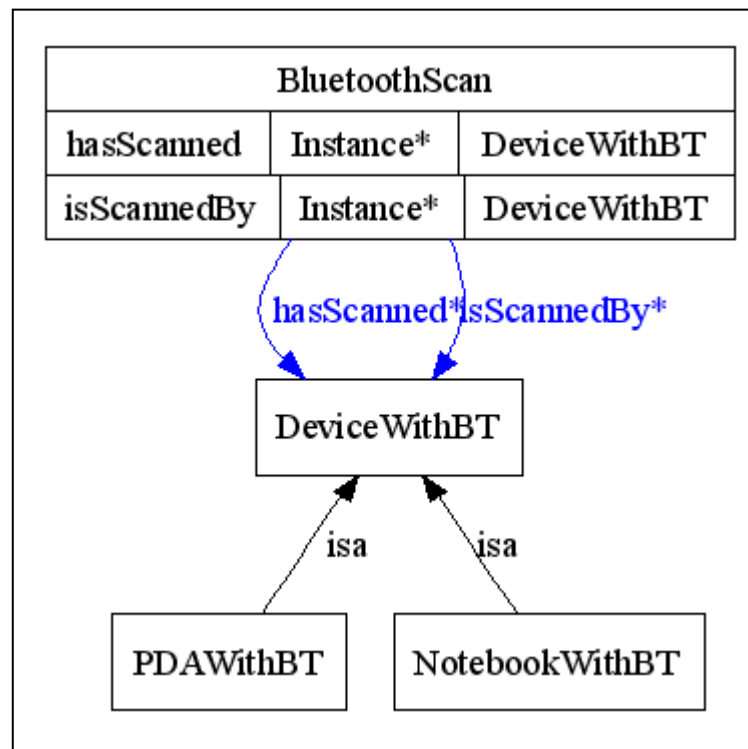


Figure 15: *BluetoothScan KnowledgeParameter* subclass

User Goal

In the SPICE knowledge layer, *Goal Deriver Knowledge Sources* attempt to infer a person’s goals from observations. Similar to the approach taken for modelling a person’s location with the reification approach, a person is represented by a *Person* concept and a goal is modelled as an instance of the *Goal* concept. Although a single *Goal Deriver* is typically able to only infer a relatively small set of goals, the diversity of possible goals that may be derived in general is quite large. Figure 16 shows a possible subset of goals that may be useful to services on the SPICE platform; searching for an entity (e.g. a restaurant or a person) and attempting to purchase something (e.g. a movie ticket). The generic *KnowledgeParameter* subclass that relates one or more goals to a person is denoted as

the *GoalLink* concept, which contains the corresponding *isKnowledgeOf* subproperties *isGoalOf* and *hasGoal*.

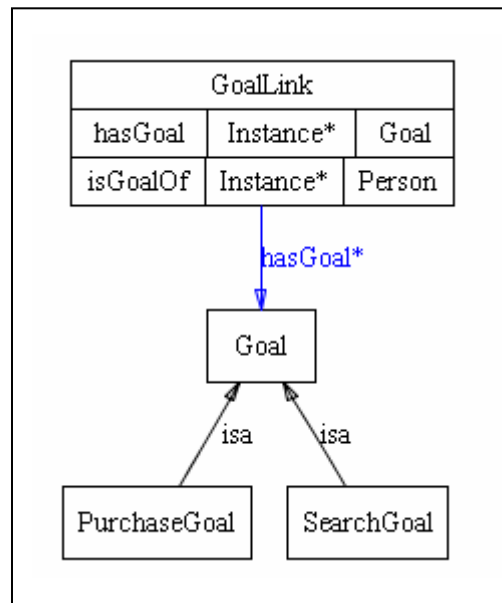


Figure 16: *GoalLink KnowledgeParameter* subclass

4.3.3 RDF++

An alternative approach is to represent relationship attributes in the A-Box only. Concepts and their relationships are described as part of the taxonomy of an ontology, generally referred to as *T-Box*. In contrast, the *A-Box* contains assertions about concept instances. In OWL instances are called *individuals*.

Figure 17 illustrates our alternative approach. Similar to the reification approach we use OWL to represent the T-Box. The major difference is that we maintain direct links between concepts. Individuals are represented in an extension to RDF which we will refer to as RDF++. RDF++ differs from RDF in that properties (OWL terminology for relationships) can be associated with additional attributes. The attributes themselves are instances of classes described in OWL.

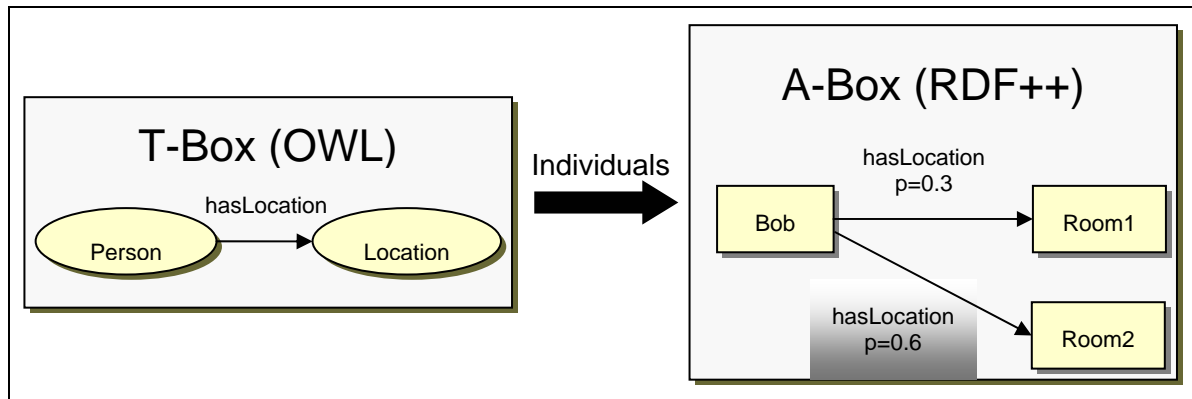


Figure 17: General approach

Figure 18 shows an example of a RDF++ document that represents the A-Box depicted in Figure 17. The highlighted parts show the definition and use of an additional property `hasConfidence`. As can be seen in the example, the `hasConfidence` element is a subelement of `hasLocation` and therefore represents a relationship attribute.

```

<owl:Class rdf:ID="Location"/>
<owl:Class rdf:ID="Person"/>
<owl:ObjectProperty rdf:ID="hasLocation">
  <rdfs:range rdf:resource="#Location"/>
  <rdfs:domain rdf:resource="#Person"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="hasConfidence">
  <rdfs:range
    rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>
<Person rdf:ID="Bob">
  <hasLocation>
    <Location rdf:ID="Room1"/>
    <hasConfidence
      rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
      0.3
    </hasConfidence>
  </hasLocation>
</Person>

```

Figure 18: Example A-Box representation in RDF++

For each relationship attribute (see Section 4.1.1) we introduce a new property. If the range of the property is more complex, an additional class is defined. For example *TemporalValidity* is a separate concept.

In initial tests Protégé was able to parse RDF++ instances by simply ignoring property attributes. However, some problems exist when properties that are modeled with attributes are edited in Protégé.

In summary, this approach has the advantage that links between the main concepts are maintained and the ontology can easily be designed in a way compatible with the SPICE Mobile ontology. More importantly, the ontology has a lower complexity in terms of number of classes and properties. Finally, existing Semantic Web tools can be used for T-Box reasoning and for limited A-Box reasoning. A disadvantage of this approach is the fact that no T-Box reasoning about relationship attributes can be performed and that existing query languages cannot easily be re-used.

4.3.4 Comparison

Both approaches have advantages and limitations, which are briefly outlined in this section.

The reification approach aims for full semantic modelling of attribute relationships and therefore includes respective concepts in the T-Box. The benefit of this approach is that QoC related information is represented in the T-Box without making any modifications to OWL/RDF. This way QoC related concepts can be treated as any other concepts and existing query languages such as SPARQL [SPARQL] can be used without restrictions.

The main limitation of the reification approach is that it introduces a higher modelling and query complexity and loses the direct links between concepts. The loss of direct links is a serious issue, as (1) domain experts may have difficulties in understanding how to semantically describe their domain and (2) may lead to compatibility and synchronization issues with other ontologies such as the SPICE Mobile ontology as concepts added to one ontology should also be present in the other ontology.

In contrast the RDF++ format aims for compatibility with existing ontologies and high usability experts and represents QoC related information only partly in the T-Box. The main benefit of this approach is that direct links are maintained and the modelling complexity in terms of the number of classes is lower. Basic compatibility with the SPARQL query language and other tools is still maintained.

A disadvantage of RDF++ is the problem that QoC related information is not fully represented in the T-Box and therefore cannot be used for T-Box reasoning. This has been a conscious design decision to maintain the usability and compatibility of the ontology while at the same time dealing with QoC related information.

The detailed differences between the two approaches are summarized in Table 2.

	Reification Approach	RDF++
Compatibility with current SPICE Mobile Ontology	Requires additional mapping	Yes
T-Box QoC Reasoning	Yes	No
Query Languages	Full use SPARQL	Limited use of SPARQL
Language modifications	None	Extension to allow property attributes
Modelling Complexity	High	Low
Direct Links between Concepts	No	Yes
Combined T-Box/A-Box reasoning	Yes	Yes
RDF/OWL parser support	Full Support	confirmed with initial tests with Protégé
Support for contradicting knowledge	Yes	Yes

Table 2: Comparison between reification approach and RDF++

Based on the presented comparison of these two approaches, further research within SPICE WP4 will investigate, which of these two approaches will be chosen for use in SPICE. It may also be possible that both approaches coexist, i.e. based on the needed component's functionality, the more beneficial approach will be used.

5 Ontology Definitions

This section includes the ontologies as introduced in Figure 2 and the related paragraph, in particular the Physical Space Ontology, Service Context Ontology, User Profile Ontology, Recommendations and Learning Ontology, and the Presence Ontology.

We will first introduce the Physical Space Ontology, since physical space (location) information is used by all WP4 enabler groups and hence is not specific to a particular WP4 enabler group. Subsequently we will introduce the enabler specific ontologies in providing requirements, state-of-the-art, description and visualisation of the SPICE approaches.

Since the presented ontologies are subject to further refinement and alignment, the ontology definition should not be considered final in the current status.

5.1 Physical Space

We modelled the physical space in a generic manner, where we made use of properties depicting certain space classes are contained in other space classes. This was the result of a case study done in the Dutch Freeband programme [Brussee2005]. At that time no existing ontology was found that resembled this. This ontology can be used for tracing mobile end-users, and objects in a building, without needing to specify distances or (more special still) coordinates.

When we use a property stating that a space X is contained in another space Y, probably the most important fact about the “contained” relation is that it is a transitive relation: if X is contained in Y and Y is contained in Z then X is contained in Z. It is also a reflexive relation: each space X is contained in itself. Containment is not the only relation of geometric importance. For example, two spaces X and Y can meet, which means there is a space Z contained in both X and Y.

5.1.1 Visualisation of Physical Space Ontology

Figure 19 depicts the *Space* class hierarchy and its properties.

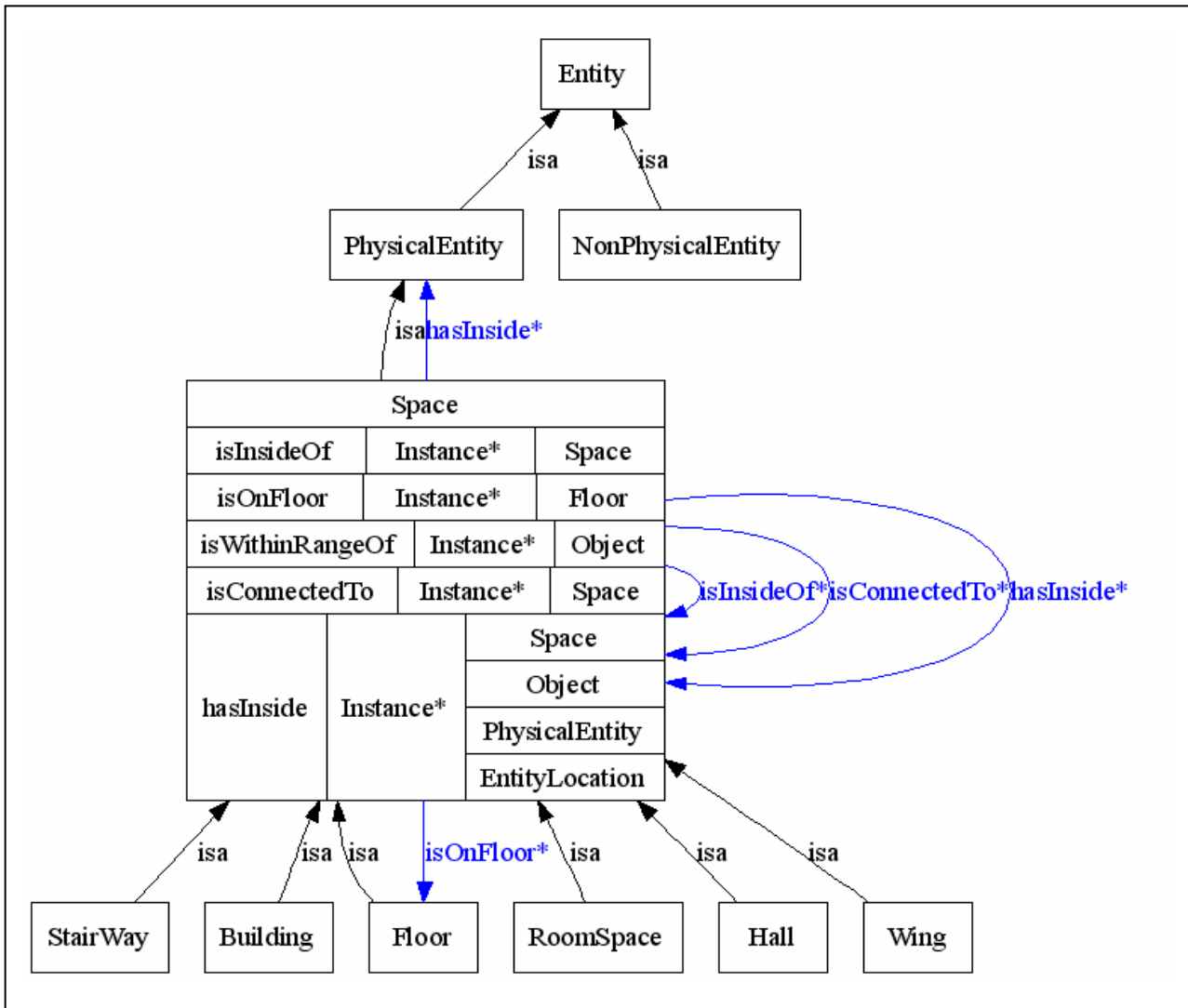


Figure 19: Illustration of (Physical) Space class and its properties

Figure 20 depicts details of the Space and Floor class and the properties *isOnFloor*, *hasAboveFloor* and *isAboveFloor* while Figure 21 depicts two instances of the Floor class.

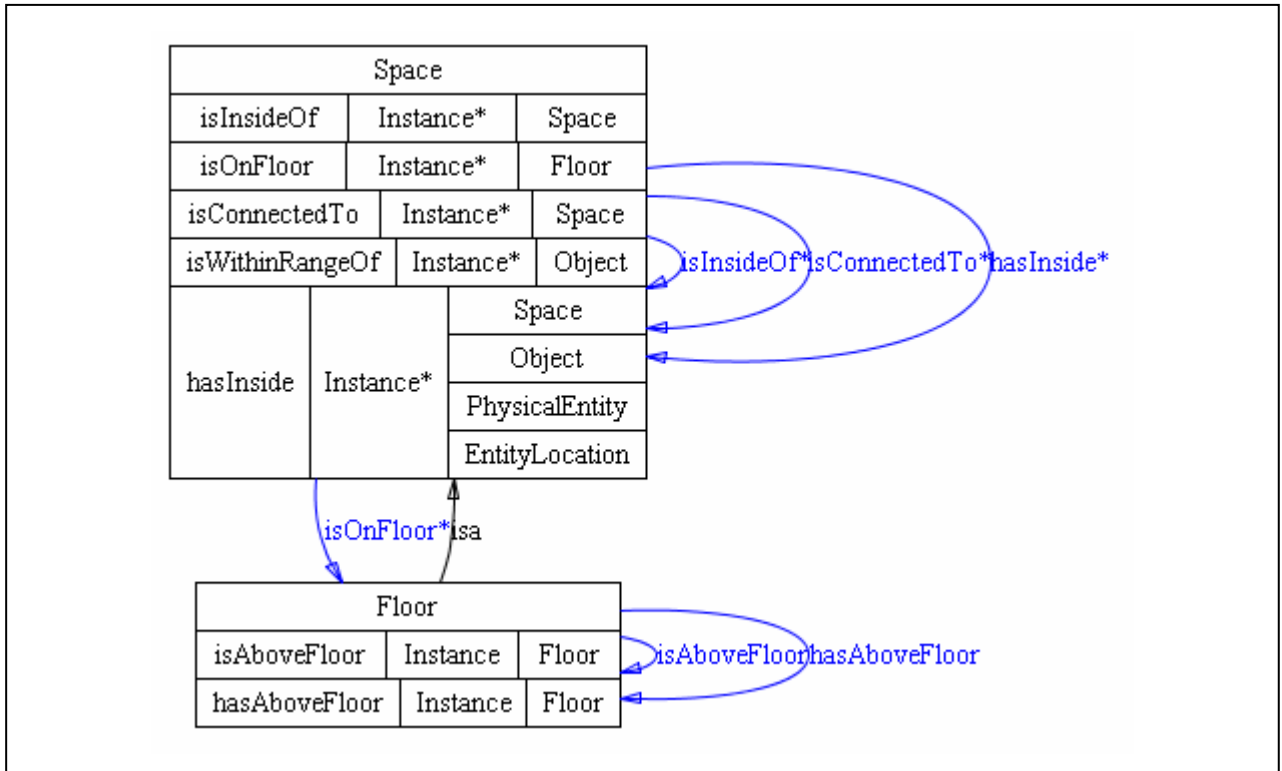


Figure 20: Illustration of properties *isOnFloor*, *hasAboveFloor* and *isAboveFloor*

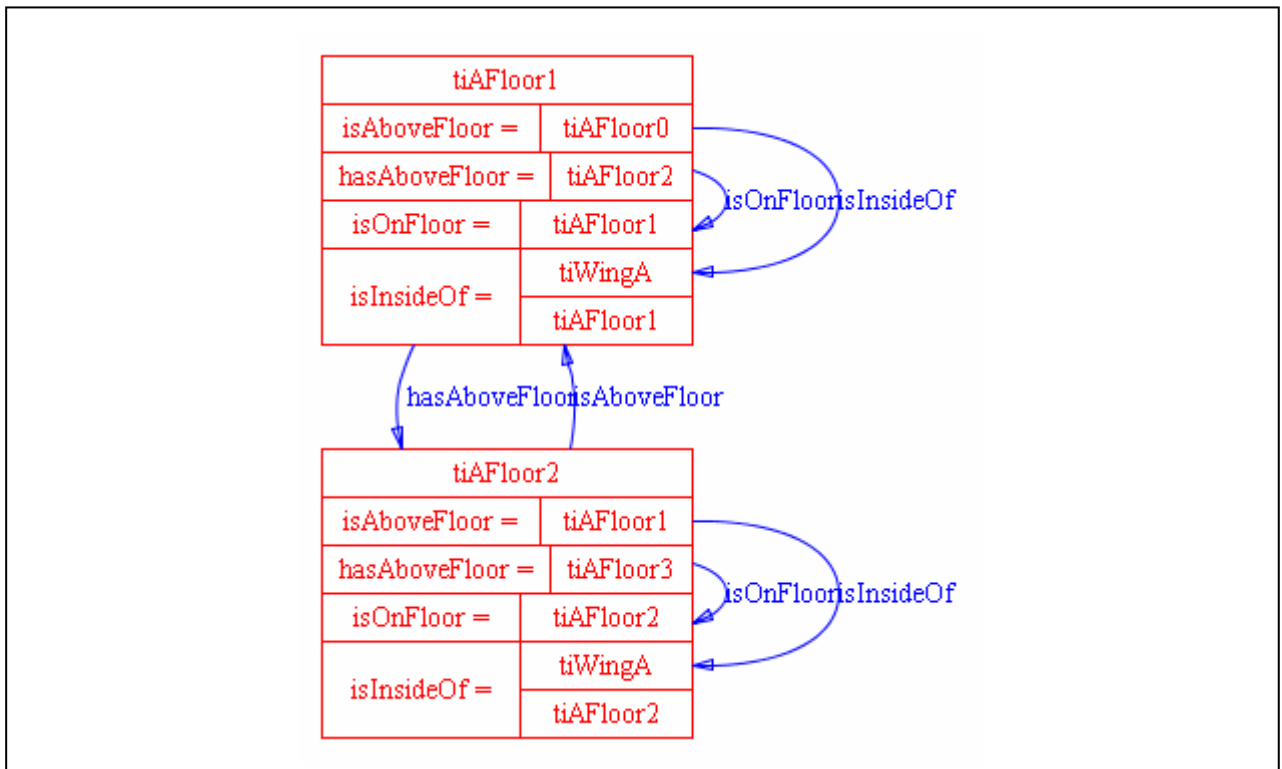


Figure 21: Illustration of instances of class *Floor*

Figure 22 depicts the relationship between the classes *Space*, *Object* and *PhysicalEntity*.

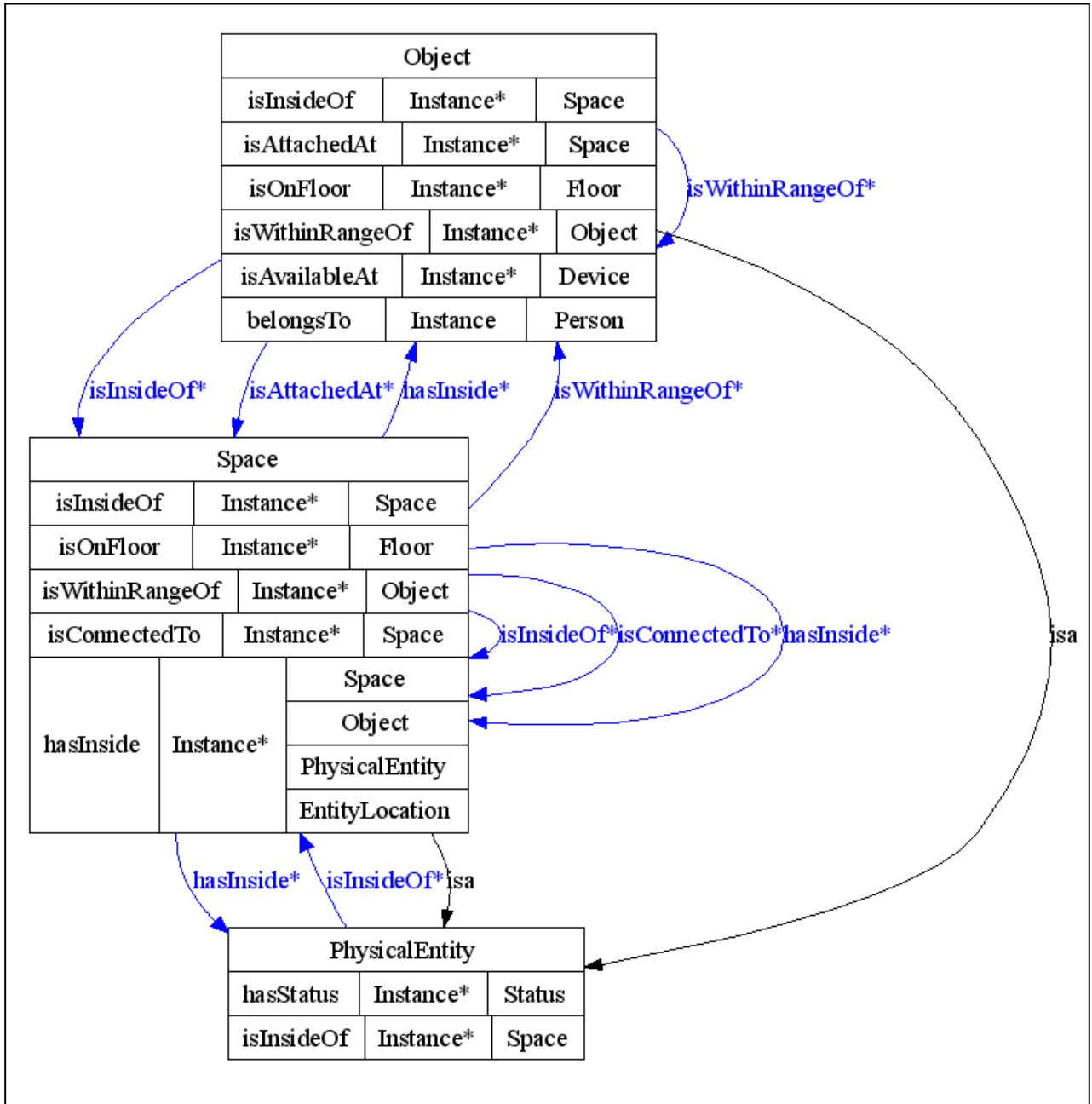


Figure 22: Illustration of *Space*, *Object* and *PhysicalEntity* classes and their properties

5.1.2 Physical Space Ontology Definition

Class: Space

Space – Any physical space, like indoors, building, room or outdoors

in-range-of: hasInside, isInsideOf, isConnectedTo, isOnFloor, isWithinRangeOf

in-domain-of: hasInside, isInsideOf, isConnectedTo, isOnFloor, isWithinRangeOf

Subclasses of Space are Class: Outdoors and Class: Indoors. Subclasses of Class: Indoors are Class: Building, Class: Roomspace, Class: Hall and Class: Floor

Property: isInsideOf

isInsideOf – is located in, such as a room is located inside a building, a Object is located within a room

Domain: Space, Object, Person

Range: Space

This property is transitive and has an inverse hasInside. This property has a subproperty isOnFloor

Property: hasInside

hasInside – has somethin inside, such as a room has a Object/terminal inside, a building has a room inside

Domain: Space

Range: Space, Object, Person

This property is transitive and has an inverse isInsideOf

Property: isOnFloor

isOnFloor - is located at a Floor, such as a room is located on a floor of a building

Domain: Space, Object, Person

Range: Space

Property: isConnectedTo

isConnectedTo – indicates adjacent spaces, such as neighbouring rooms or an elevator next to a floor.

Domain: Space

Range: Space

This property is transitive and symmetric. This property has a subproperty *isDirectlyConnectedTo*

Property: hasAboveFloor

hasAboveFloor – indicating the existence of above floors, such as can be useful in a multi-floor building

Domain: Floor

Range: Floor

This property is functional and inverse-functional and has an inverse *isAboveFloor*

Property: isAboveFloor

isAboveFloor – indicating the existence of floors below, such as can be useful in a multi-floor building

Domain: Floor

Range: Floor

This property is functional and inverse-functional and has an inverse *hasAboveFloor*

Property: isWithinRangeOf

isWithinRangeOf - is within range of eg a gsm-cell, or Wi-Fi network

Domain: Space, Object

Range: Object

This property is transitive

5.1.2.1 Additional to Inherited Ones from Space

Class: Stairway

Stairway - indicating any staircase

in-range-of: hasAboveFloor, isAboveFloor

in-domain-of: hasAboveFloor, isAboveFloor

Class: Object

Object – Any Object, such as terminals, screens, schedule software, sensors, access networks, etcetera.

in-range-of: isInsideOf, isOnFloor, isWithinRangeOf, isAvailableAt, belongsTo

in-domain-of: isInsideOf, isOnFloor, isWithinRangeOf, isAvailableAt, belongsTo

Property: belongsTo

belongsTo - indicating the owner, such a the owner of a device

Domain: Object

Range: Person

This property is funcitonal and has an inverse owns

Property: owns

owns - indicating the owner, such a the owner of a device

Domain: Person

Range: Object

This property is inverse_funcitonal and has an inverse belongsTo

Property: isAvailableAt

isAvailableAt - indicating if a device is equipped with certain capabilities, such as AN IF

Domain: Object

Range: Device

This property is transitive and has an inverse *isEquippedWith*

Property: isEquippedWith

isEquippedWith - indicating if a device is equipped with certain capabilities, such as AN IF

Domain: Device

Range: Object

This property is transitive and has an inverse *isAvailableAt*

5.2 Service Context

Network nodes, users and services are all entities relevant for service delivery. User context has been addressed in numerous projects and papers. And most often, when the term context is used alone, it actually refers to user context. However, all entities can have context. While user context is information on a user's whereabouts (location, mood, current doings, etc.), Service Context is information on a service's current capabilities (available QoS, when/where is the service valid – and for whom, etc.). Furthermore, Service Context is related to service metadata much in the same way user context is related to user profiles. That is, the static information on a service is kept in the service metadata, and the dynamic aspects are defined to be Service Context.

In order to enable brokering and other kinds of interoperation with the Service Context, it is organized in an ontological manner. The Service Context ontology is modelled using both UML and OWL. This is to provide multiple views on the ontology: the UML model gives an overview of organisation and architecture, the OWL model shows details on the classes and relations of the ontology.

The SPICE Service Context ontology will be developed in parallel with the SPICE architecture. The efforts presented in this document are directed towards creating the skeleton for the Service Context ontology modelling.

Before specifying the Service Context ontology, the following points should be discussed:

- Usage of the Service Context information in the SPICE platform
- Organisation of the Service Context information
- Which information belongs to the Service Context

5.2.1 Usage of Service Context Information in the SPICE Platform

The first question to be answered before focusing on the Service Context ontology is how service context information will be used in the SPICE service platform and by whom. Figure 23 illustrates that some SPICE service components might interact directly with the knowledge architecture layer (push and pull techniques – interacting directly with the *SKPN* components). Figure 24 illustrates another approach, by which the SPICE architecture components act directly as knowledge (i.e. service context) consumers and interact directly with the *Knowledge Sources*. Service context (and its ontology) should be modelled and designed in order to support both ways of interaction.

They require ontology design compromises, which can be exemplified by the following:

- centralised ontology vs. decentralised shared ontology
- richness in ontology classes and their attributes (slots) vs. richness in ontology relations (properties)
- centralised vs. decentralised ontology processing

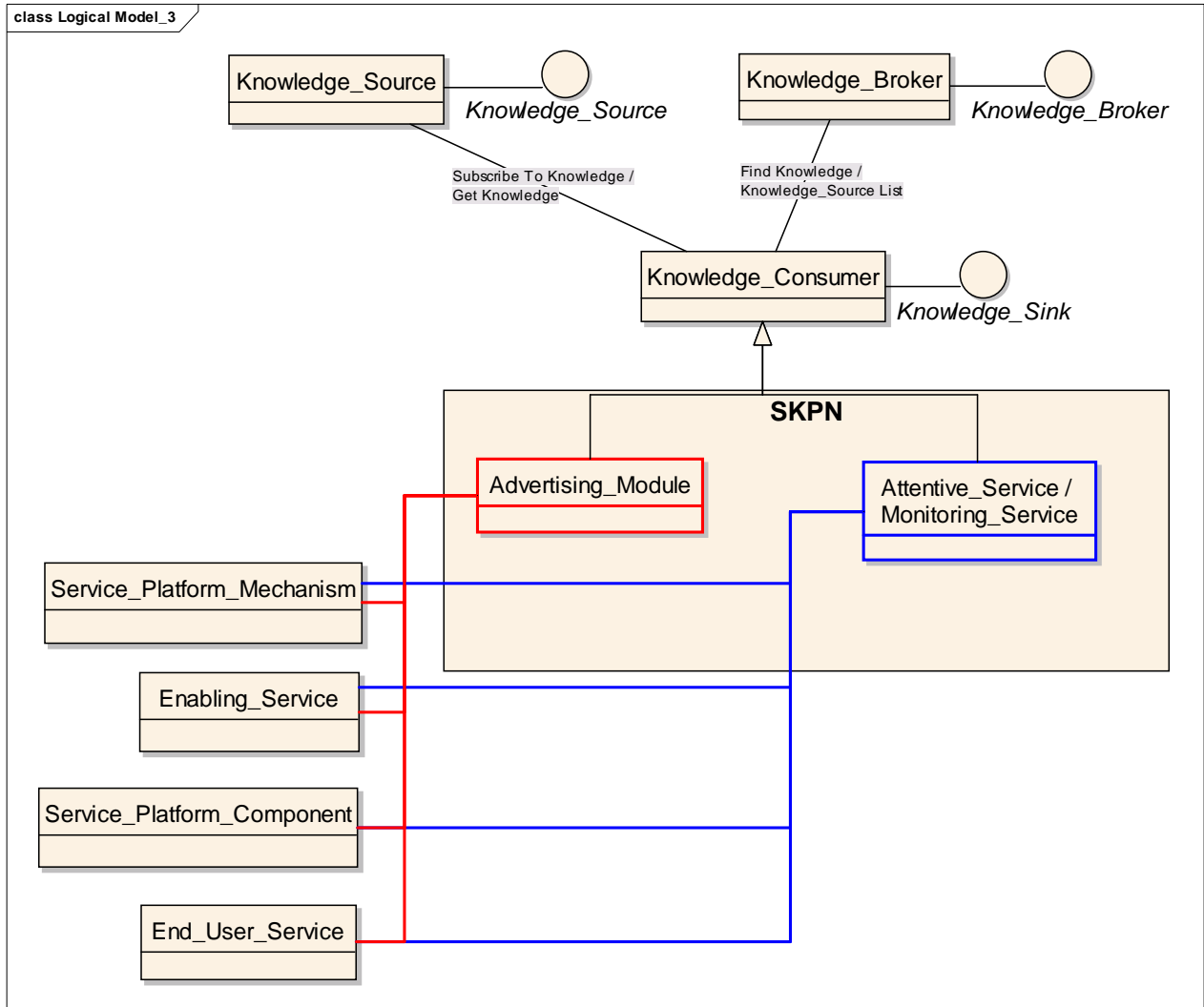


Figure 23: Usage of service context information: several alternatives can be implemented, e.g. via context monitoring service (blue), or advertising module (red) (SKPN components)

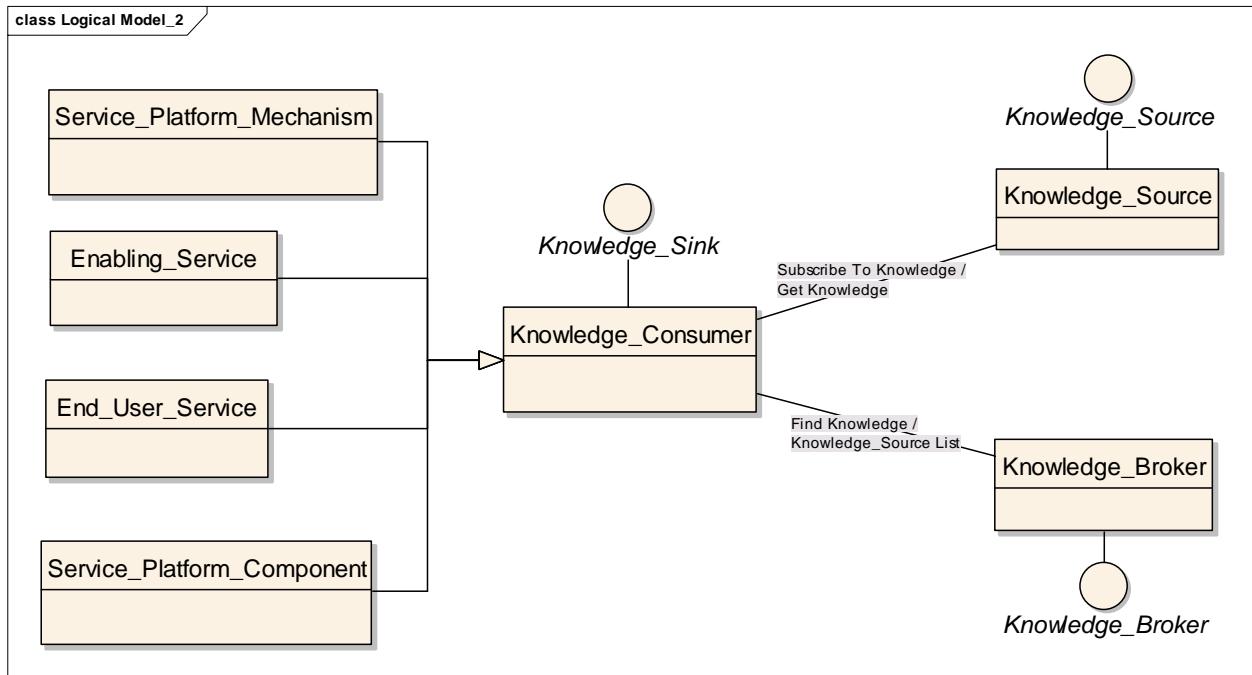


Figure 24: Direct usage of the service context (as knowledge consumers)

5.2.2 Organisation of Service Context Information

The SPICE architecture design principle leans on the loosely coupled service components. It allows:

- Coexistence of several solutions for the service context retrieval and management
- Variety of service context elements and containers
- Difference in the way how the context is used and processed
- Distributed logic for ontology processing
- Compromise of several context retrieval and management solutions

The SPICE service architecture is in design phase, and the service ontology has to be modelled / designed in parallel to other design activities. This uncertainty requires some modelling and design assumptions, in order to start ontology modelling work. Figure 25 illustrates the initial draft design assumptions about the overall organisation of the service context information. This structure is subject to changes and updates and it should be understood as the draft material. A detailed visualisation and description of the service context ontology related classes can be found in 5.2.3 and 5.2.4.

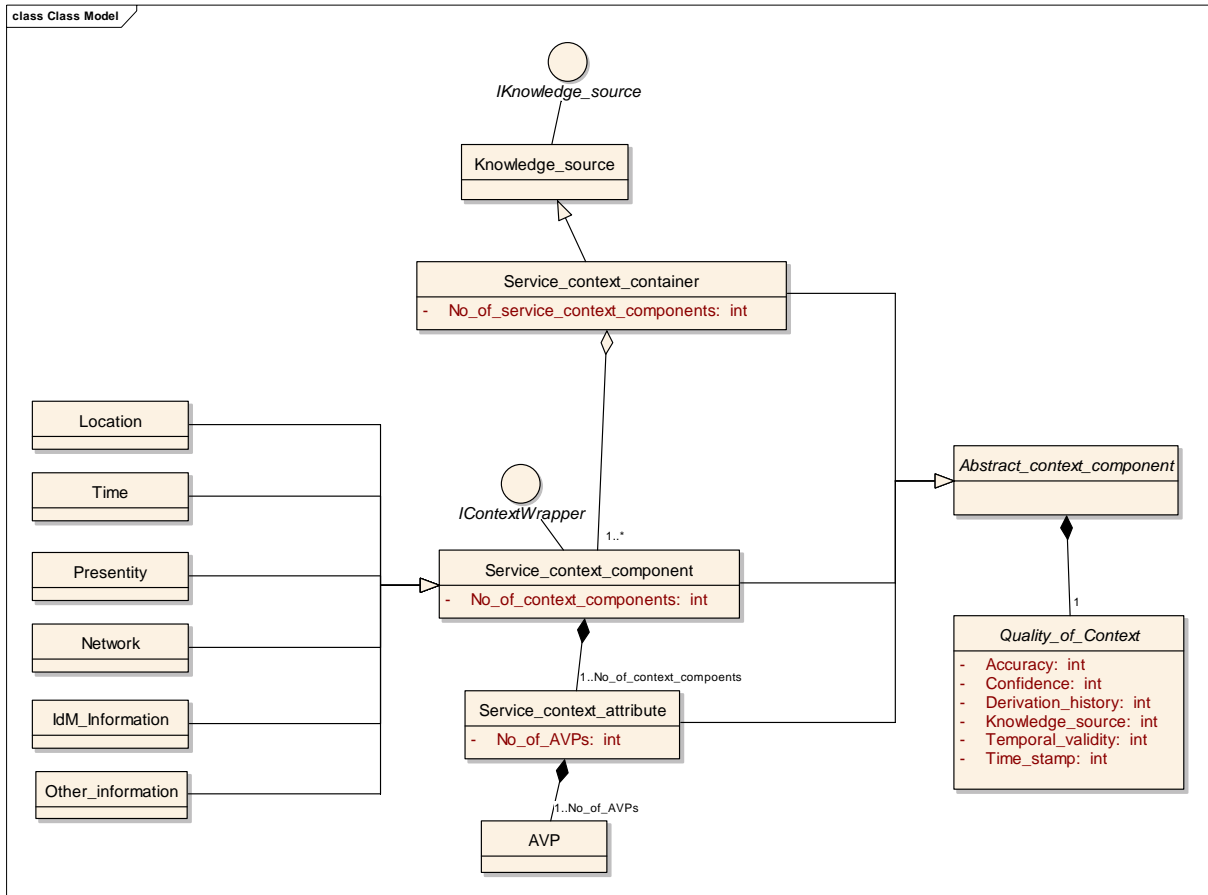


Figure 25: Option in the organisation of the service context information

5.2.3 Visualisation of Service Context Ontology

Figure 26 shows the top-level class hierarchy for the Service Context ontology. The ontology is meant to reflect the UML model in section 5.2.2.

A number of classes that are part of other ontologies are referenced in the Service Context ontology. Examples are *QualityOfContext* and *Location*. The relations with other ontologies – both SPICE-specific and external ones – will be detailed in forthcoming versions of the Service Context ontology.

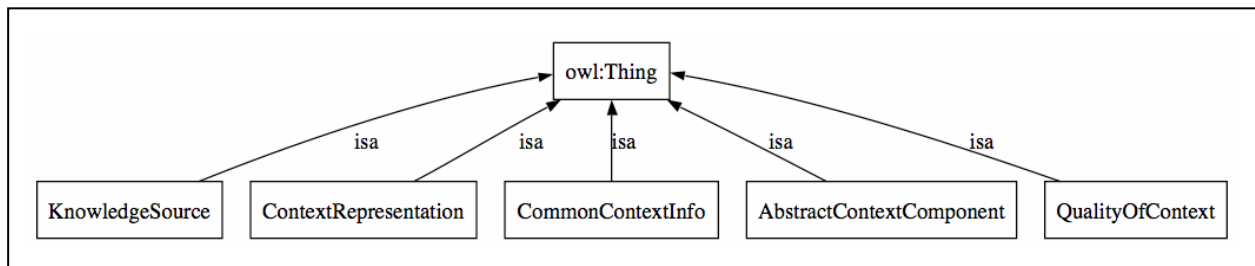


Figure 26: Top-level class hierarchy for service context

Figure 27 depicts the sub-tree of the *ContextRepresentation* class that describes various representations of an object's context, e.g. Attribute Value Pair (AVP).

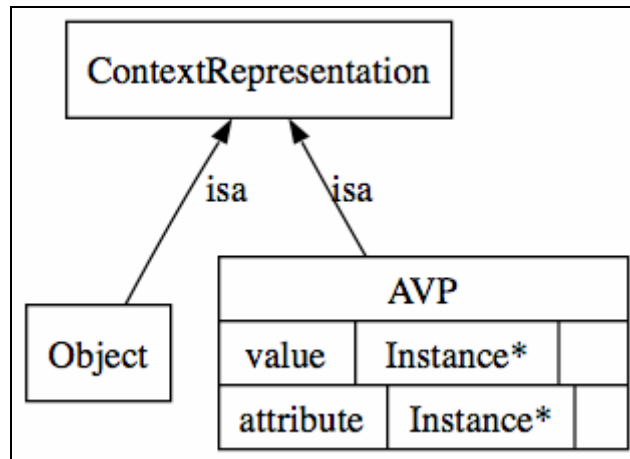


Figure 27: *ContextRepresentation* class and subclasses

Figure 28 depicts the sub-tree of the *CommonContextInfo* class that holds common information about context, e.g. how valid / trustworthy is the context information at hand.

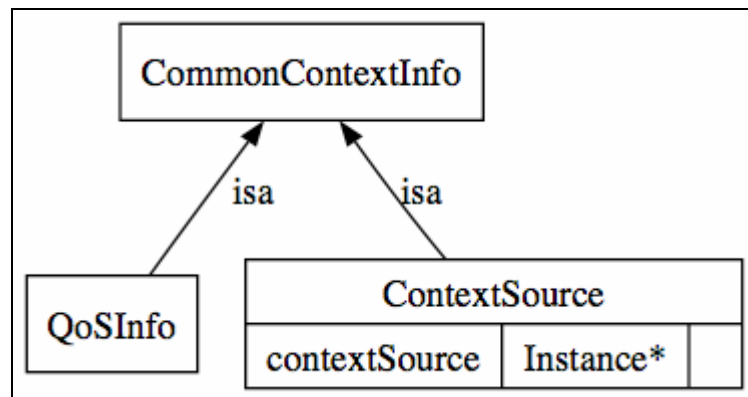


Figure 28: *CommonContextInfo* class and subclasses

Figure 29 depicts the *AbstractContextComponent* class, its properties and its sub-tree. The *AbstractContextComponent* class is a superclass of three kinds of contexts information classes that are identified. In Figure 30, the sub-tree of the *ServiceContextElement* class (one of the subclasses of *AbstractContextComponent*) is depicted in more detail.

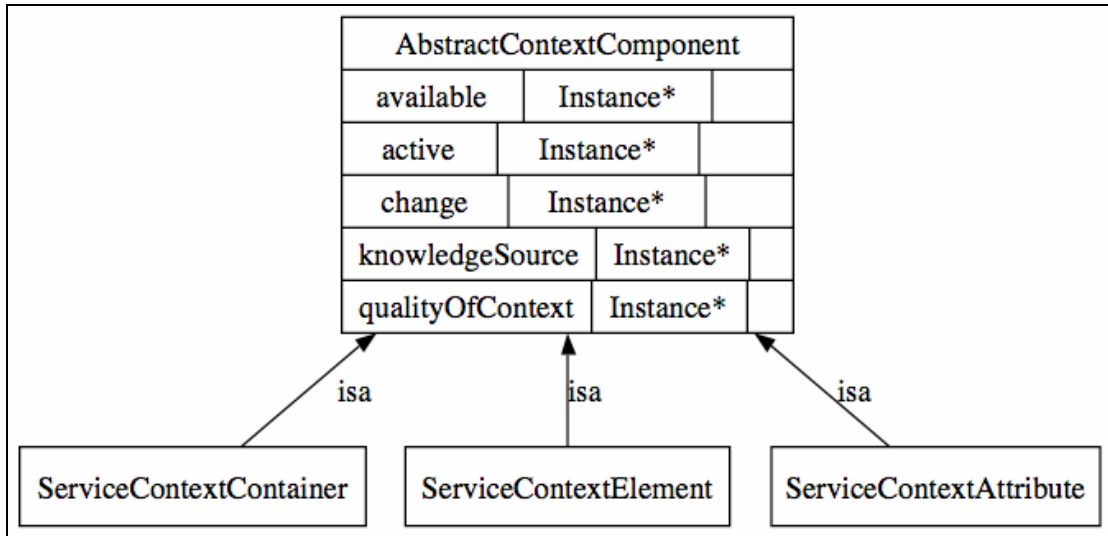


Figure 29: *AbstractContextComponent* class and subclasses

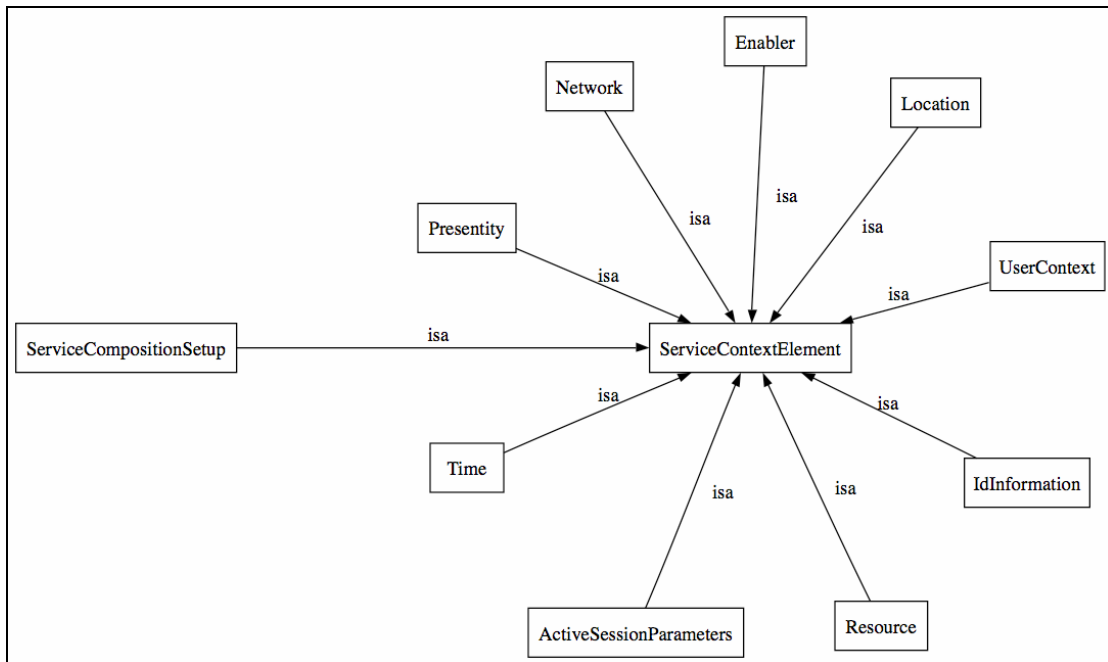


Figure 30: *ServiceContextElement* class and subclasses

5.2.4 Service Context Ontology Definition

Class: ContextRepresentation

ContextRepresentation – A Context Representation

The `ContextRepresentation` class holds various representations of an object's context, e.g. Attribute Value Pair (AVP).

The `Object` class contains the context attribute representation - in some format and is a subclass of the `ContextRepresentation` class.

The `AVP` class represents the Attribute Value Pair (AVP) representation of context information and is a subclass of the `ContextRepresentation` class.

Class: CommonContextInfo

CommonContextInfo – Common information about context

The `CommonContextInfo` class holds common information about context, e.g. how valid / trustworthy is the context information at hand.

The `QoSInfo` class contains general information on QoS related to a service, e.g. currently available QoS, and is a subclass of the `CommonContextInfo` class.

The `ContextSource` class is shared between all ontologies on context (user, service, etc.). It keeps information on the source of the context information. Based on this the user of the context information can get a view of how good the information is; e.g. does the context information come from an accurate source or not. The `ContextSource` class is a subclass of the `CommonContextInfo` class.

Class: KnowledgeSource

KnowledgeSource – A Knowledge Source

The `KnowledgeSource` class represents a root concept for *Knowledge Sources*. This class is kept as a reference, and points to an ontology that copes with *KMF* specific class definitions.

Class: QualityOfContext

QualityOfContext – A Quality of Context

The `QualityOfContext` class represents the root concept Quality of Context (QoC). This class is kept as a reference, and points to an ontology that copes with QoC specific class definitions.

Class: AbstractContextComponents

AbstractContextComponents – An abstract class containing three identified kinds of context information

The `AbstractContextComponents` class is an abstract class. Within this class there are three kinds of context information that are identified. The `Property:Active` indicates part of the active context, e.g. a UMTS network is available. The `Property:Available` indicates elements that are part of a service's context, but are not active, e.g. a Bluetooth network can be available but not used. The `Property:Change` indicates changes in environments that can influence the context, e.g. a new network becomes available. Together these three types can give an overview of the current service context (active), what might be (available), and the changes in context (change).

Class: ServiceContextContainer

ServiceContextContainer – A service context container

The `ServiceContextContainer` class represents a collection of context elements and is a subclass of the `AbstractContextComponents` class.

Class: ServiceContextAttribute

ServiceContextAttribute – A service context attribute

The `ServiceContextAttribute` class represents an isolated piece of context information, e.g. location information and is a subclass of the `AbstractContextComponents` class.

Class: ServiceContextElement

ServiceContextElement – A service context element

The `ServiceContextElement` class is a super class for service context elements. Service context elements are holders of specific service context information. Examples are location and network information (in a service context context - no pun intended). This class is a subclass of the `AbstractContextComponents` class.

The `Presentity` class represents information on presentity and represents a reference point to the dedicated ontology for presentity. This class is a subclass of the `ServiceContextElement` class.

The `Network` class represents information on a network and represents a reference point to a separate ontology for network resources. This class is a subclass of the `AbstractContextComponents` class.

The `Enabler` class represents information on enablers and is a subclass of the `AbstractContextComponents` class.

The `Location` class represents information on location and represents a reference point to a separate location ontology. This class is a subclass of the `AbstractContextComponents` class.

The `UserContext` class represents information on user context and represents a reference point to a specific user context ontology. This class is a subclass of the `AbstractContextComponents` class.

The `IdInformation` class represents information on Ids and is a subclass of the `AbstractContextComponents` class.

The `Resource` class represents information on resources. This class is a subclass of the `AbstractContextComponents` class. This class is a subclass of the `AbstractContextComponents` class.

The `ActiveSessionParameters` class represents information on active session parameters and is a subclass of the `AbstractContextComponents` class.

The `Time` class represents information on time and represents a reference point to an overall Time ontology. This class is a subclass of the `AbstractContextComponents` class.

The `ServiceCompositionSetup` class represents information on service composition setup and is a subclass of the `AbstractContextComponents` class.

5.3 User Profiles

This section includes the requirements, state-of-the-art and SPICE WP4 ontology approach for the Personal Information Enabler related information. In particular, this section deals with the specification of the SPICE user profile ontology.

5.3.1 Requirements

One of the principal objectives of SPICE is to provide users a whole pallet of services according to their situations and tastes, i.e. preferences. For this purpose, miscellaneous user information will be collected and structured into user profiles. These profiles offer the advantage of being easily enriched and exploitable by SPICE platform services in order to deliver to the user, at any moment, the best fitted service with regard to his situation.

The requested functionality of the SPICE service platform requires means for managing user profiles of different SPICE platform services and services that are external to the SPICE platform. Thereby, we have to deal with two major requirements. Firstly, we have to provide means for describing, managing and providing context-dependent user information and user preferences for the contextual personalisation of services, i.e. the service personalisation that takes into account the user's current context.

Secondly, we have to consider that different services, in particular SPICE platform services and services that are external to SPICE, may use different semantics for describing user data. Since we do not want to impose specific user data related semantics upon service developers, a user profile structure is needed that does not restrict a service developer in the development and use of specific user data vocabularies.

In the following, we analyse the most promising existing user profile representations and schemas with regard to their support for these requirements. Subsequently, we present our approach for a user profile schema that leads to the definition of our user profile ontology.

5.3.2 State of the Art

A lot of work has been done to propose user profile approaches, representations and schemas. A very interesting approach is described in the **Doppelgänger User Modelling System** [Orwant1995]. In this approach, each user model consists of submodels, in particular domain submodels and conditional submodels. Domain submodels contain information about a particular aspect of the user's behaviour such as his/her preferences for the retrieval of sports news. Conditional submodels on the other hand contain information about the user when a certain situation arises such as a specific time of day or a specific user activity. In this respect, this approach seems very useful for the SPICE objectives. On the one hand domain submodels can be used in order to represent user models that are specific to a particular SPICE service. For instance, there could be one domain submodel for a news feed service and one domain submodel for the telephony service. Furthermore, there could be different conditional submodels for each service containing situation-specific preferences. For instance, there could be different conditional submodel for the news feed service, one for the situation *user is at home*, one for the situation *user is in his office* and one for the situation *user is driving a car*. Unfortunately,

the Doppelgänger approach is described on a conceptual level without specifying a concrete vocabulary for a user profile structure.

The W3C **Composite Capabilities/Preference Profiles** (CC/PP) [CCPP], [Suryanarayan2002] defines a structure and vocabulary for describing device capabilities and user preferences that can be used to guide the adaptation of content presented to that device. It provides a two-level-hierarchical profile structure where a profile has one or more components and each component has one or more attributes. Default profiles are supported as well as extensibility with application specific vocabulary. On the other hand, CC/PP does not consider contextual constraints in order to express that some preferences are only valid in a specific context.

UAProf [UAProf], which can be seen as an example application of CC/PP focuses on hardware and software characteristics of devices instead of user preferences profiles. Beside this, it has the same shortcomings as CC/PP.

The 3GPP **Generic User Profile** (GUP) [GUP] defines a user profile structure and data description method where a profile is defined as a collection of profile components. A profile component is defined as a collection of data elements, also called data payloads. Furthermore, GUP defines a top-level schema for profile descriptions. GUP profile components consist of a description which can be used for usage properties. However, the specification does not provide details on how usage properties may look like and how they may be used, e.g. for the description of contextual constraints.

The **Open Travel Alliance** (OTA) [OTA] defines a specification of messages of requests and answers, used daily in the trade of voyages. Generic diagrams in XML have been defined for the messages exchanged in scenarios of reservation of planes, cars, trains, hotels, organised voyages, and also to get information on a travel route. The OTA also wants to standardize the profile of a traveller. The profile of an OTA traveller contains basic information, such as the identification of the traveller, or of the company, critical information to the financial transactions, co-ordinates etc. However, the OTA Profile is specific to the trade of voyages and hence not sufficient for the SPICE purposes.

The **FOAF** [FOAF] project is based around the use of machine readable Web homepages for people, groups, companies and other kinds of thing. To achieve this, the "FOAF vocabulary" is used. It provides a collection of basic terms that can be used in these Web pages. At the heart of the FOAF project is a set of definitions designed to serve as a dictionary of terms that can be used to express claims about the world. The FOAF Vocabulary definitions presented here are written using RDF/OWL. However, FOAF does not consider different sub-profiles for different user situations.

5.3.3 Approach

Our approach for a SPICE user profile structure considers the creation of different sub-profiles for different SPICE services and enables the inclusion of contextual constraints that specify specific usage conditions. That is, our approach basically follows the Doppelgänger User Modelling approach [Orwant1995], since it seems to fit best the SPICE objectives, as explained in 5.3.2. First, we will introduce this user profile structure. Subsequently we will depict the resulting user profile ontology.

The overall high level user profile structure is depicted in Figure 31. The user profile is separated into different subsections that specify user information and preferences regarding different services. Each service-related section includes at least one sub-profile, which we call *Profile Subset* in the following. *Profile Subsets* can either include default user data or user data that is valid when a specific situation arises.

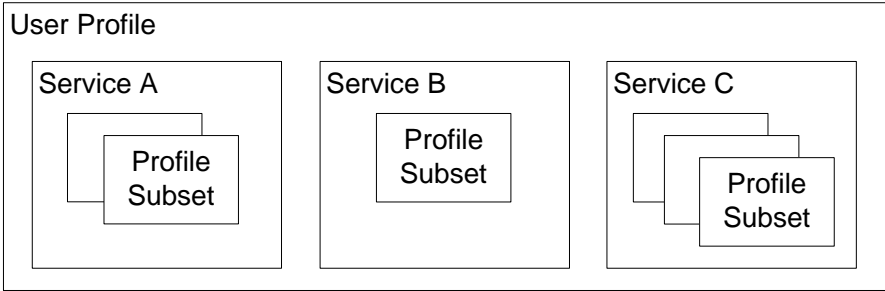


Figure 31: High level user profile structure

In the following, we briefly introduce the elements of the SPICE user profile structure. Figure 32 depicts the elements and their dependencies.

- Profile

A profile contains all persistent data about a user who is identified by a unique *entityID*. A profile can contain *Profile Subsets* that include user data with regard to different services. Since we do not want to restrict the profile structure for users only, the profile class includes an *entityType* parameter. That is, the profile structure can be used for different entity types such as users, user groups or other entities.

- Profile Subset

A *Profile Subset* contains a service-related (application-related) subset of all persistent user data. For the identification of the related service (application), it includes a unique *serviceID*. Beside the actual user data (user model), it can also contain metadata in particular conditions. A user can have several *Profile Subsets* related to the same service. These can be distinguished into *Default* (*subsetType=default*) and *Conditional Profile Subsets* (*subsetType=conditional*). However, there must be exactly one *Default Profile Subset* for each supported service that includes default user data (the default user model). In this *Default Profile Subset*, no conditions can be specified. Furthermore, there can be several *Conditional Profile Subsets* related to each service that include constraints describing the contexts in which the user data shall be applied. For instance, the constraints could express a particular time of day, a user activity or a combination (conjunction) of both.

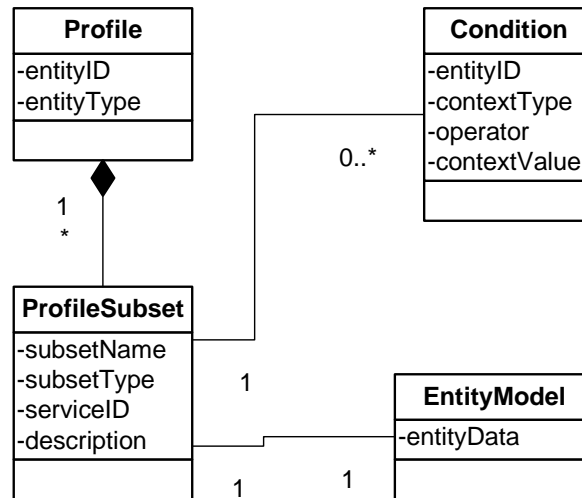


Figure 32: User profile structure including *Conditional Profile Subsets*

- Condition

The condition (or conjunction of several conditions) specifies in which situation the user data (user model) shall be applied. The condition includes the *contextType*, e.g. location, the *contextValue*, e.g. at home, the relation *operator*, e.g. equal (meaning: “if *contextValue* equals the current context of the user”) or not equal, and the ID of the related entity. The entity could be the owner of the user profile but also another person, e.g. a family member. In case the condition is empty, the *Profile Subset* represents a *Default Profile Subset*, otherwise a *Conditional Profile Subset*.

- User Model (User Data)

The user model includes the actual user data. The vocabulary for this part can be service-specific. Since we focus on a top-level user profile structure and ontology rather than on specifying the detailed content (user information) of a user profile, we do not specify this part here.

The profile structure that is depicted in UML notation in Figure 32 now has to be transformed into an ontology definition. As ontology language we chose OWL. A visualisation of the resulting ontology definition is depicted in Figure 33 and Figure 34. A user profile example instance is shown in Figure 35. The detailed class and property definitions are described in section 5.3.5.

5.3.4 Visualisation of User Profile Ontology

Figure 33 depicts the relations between *Entities*, *Profiles*, *Profile Subsets*, *Services* and *Entity Models*. In particular, an *Entity* (a user or user group) is described by exactly one *Profile*. That can consist of several *Profile Subsets*, where each *Profile Subset* is associated with a particular *Service* and consists of the service-related *Entity Model*. A

Profile Subset is either a *Default* or a *Conditional Profile Subset*. The latter includes conditions, which is shown in detail in Figure 34.

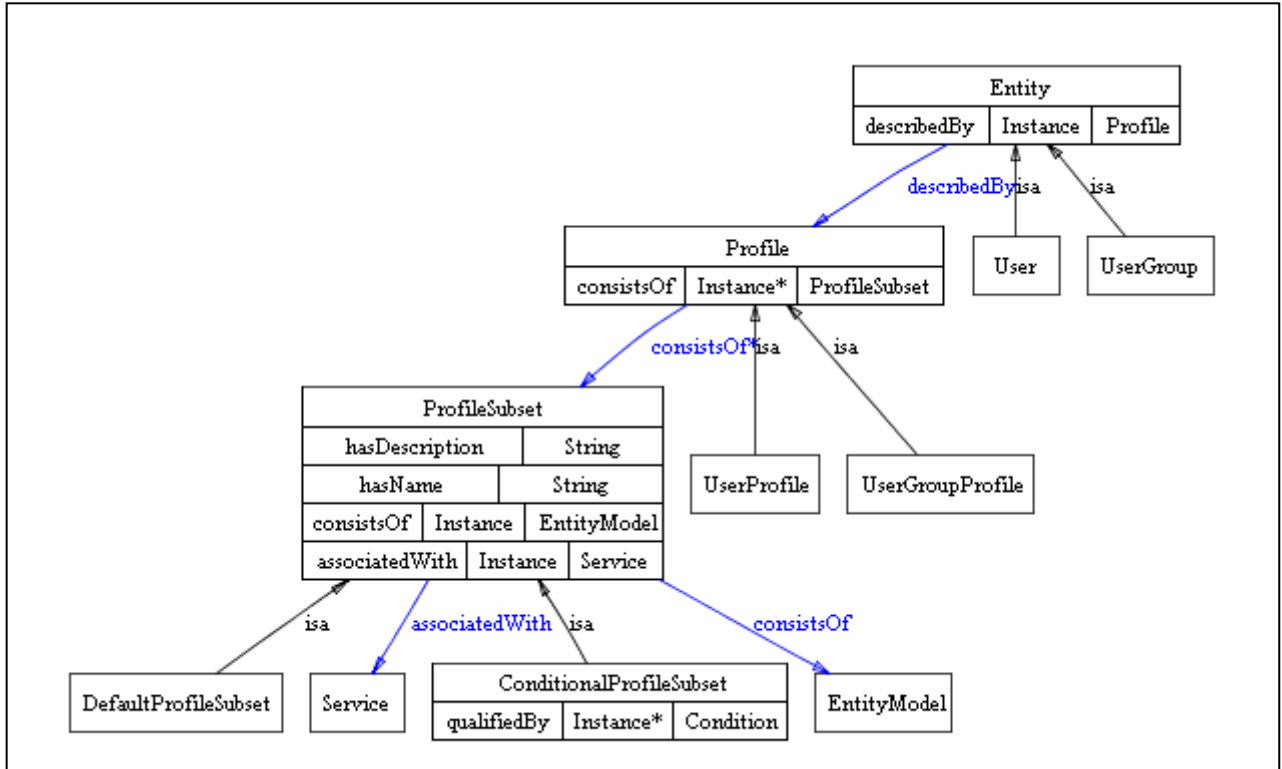


Figure 33: Visualisation of profile ontology (part 1)

Figure 34 depicts the relation between a *Conditional Profile Subset*, *Conditions*, *Context* and an *Entity*. In particular, a *Conditional Profile Subset* is qualified by at least one *Condition*. A *Condition* consists of a *Context*, such as a location or time of day that is related to a particular *Entity*, e.g. to the user himself who is described by the *Profile*. It should be mentioned that we do not aim to specify an own *Context* ontology, instead we will use context-related concepts (location, time etc.) that are specified in other parts of the SPICE Mobile ontology.

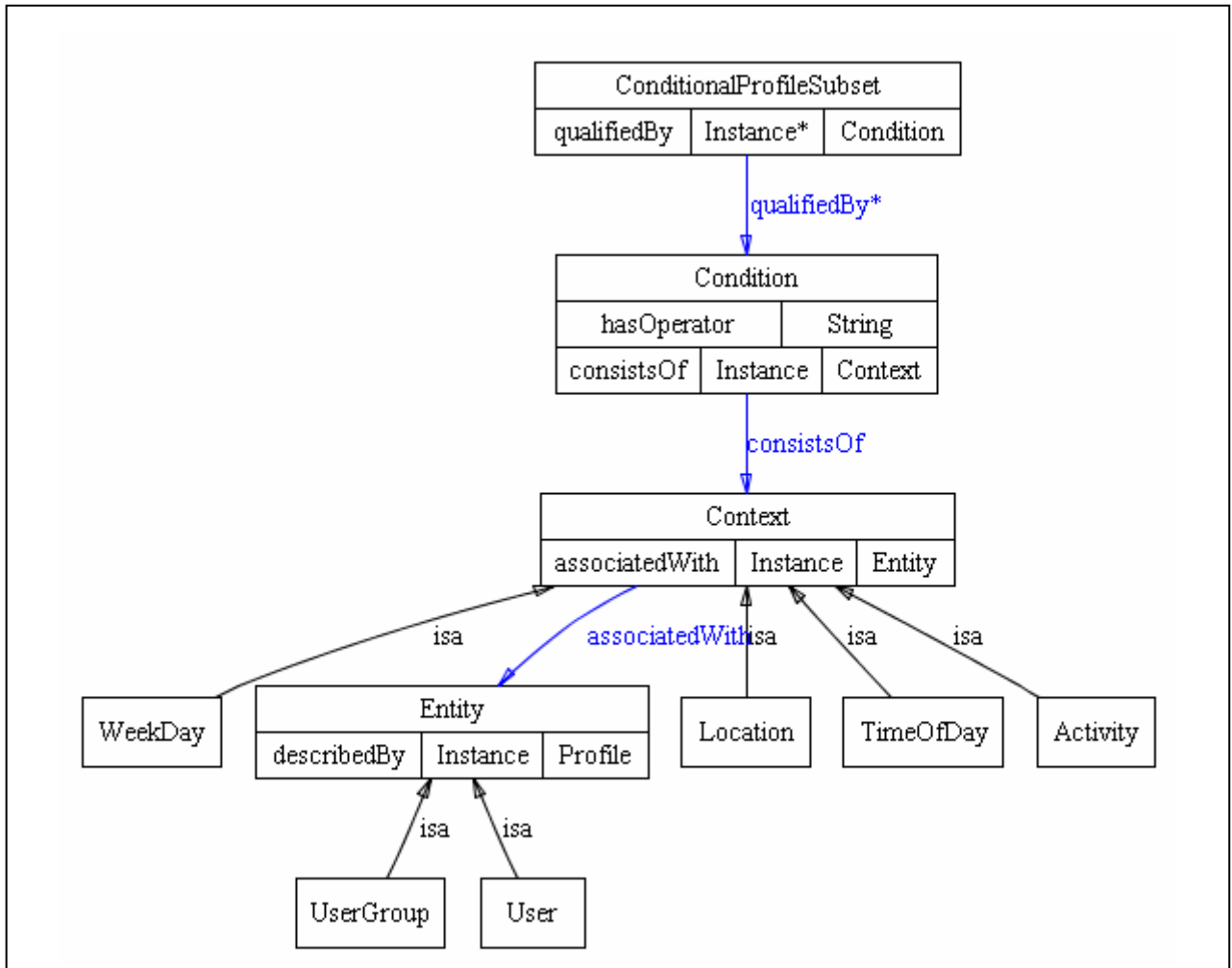


Figure 34: Visualisation of profile ontology (part 2)

Figure 35 depicts an example user instance together with an example profile instance. The user *Bob* is described by the profile *Bob_UserProfile* that consists of three *Profile Subsets*, of which only one is depicted in the figure. The depicted *Profile Subset* is a *Conditional Profile Subset* related to a service called *NewsFeedApp* and consists of one condition (*Bob_Home_Condition*). The depicted condition can be read as “this profile subset shall be applied if the context of user *Bob* equals the home location of *Bob* (*Bobs_Home*)”. In this case the user model (*Bob_Model2*) includes the actual user data and user preferences to be applied in this user context.

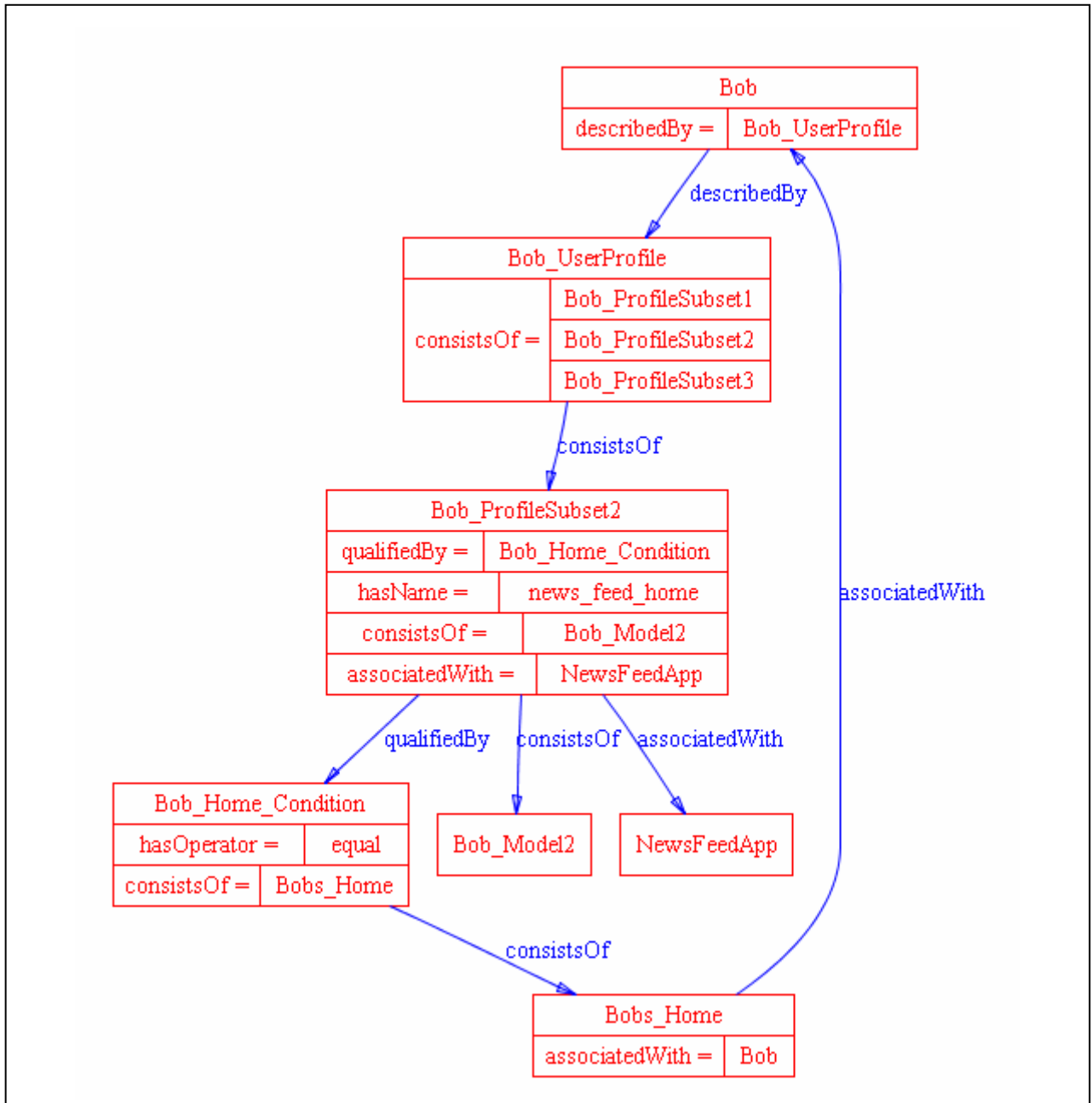


Figure 35: Profile Subset instance of user Bob

5.3.5 User Profile Ontology Definition

The following lines depict the classes and properties for the described profile ontology. Some of the used object properties are not depicted here, since they have already been defined in the DCS part of the SPICE Mobile Ontology, see [SPICE D3.1].

Class: dcs:Entity

Entity – An entity, e.g. a user or a user group

in-range-of: dcs:associatedWith

in-domain-of: dcs:describedBy

The `dcs:Entity` class represents an entity, e.g. a user or a user group.

The `dcs:User` class represents a user and is a subclass of the `dcs:Entity` class.

The `dcs:UserGroup` class represents a user group and is a subclass of the `dcs:Entity` class.

Class: dcs:Profile

Profile – An entity profile, e.g. a user profile or a user group profile

in-range-of: dcs:describedBy

in-domain-of: dcs:consistsOf

The `dcs:Profile` class represents an entity profile, e.g. a user profile, user group profile or a profile of another entity.

The `dcs:UserProfile` class represents a user profile and is a subclass of the `dcs:Profile` class.

The `dcs:UserGroupProfile` class represents a user group profile and is a subclass of the `dcs:Profile` class.

Class: dcs:ProfileSubset

ProfileSubset – A service-related subset of an entity profile, either a default profile subset or a conditional profile subset

in-range-of: dcs:consistsOf

in-domain-of: dcs:associatedWith, dcs:consistsOf, dcs:qualifiedBy

The `dc:ProfileSubset` class represents a service-related subset of an entity profile. A subset can include a default entity model or an entity model for a specific situation.

The `dc:DefaultProfileSubset` class represents a service-related default subset of an entity profile. A default profile subset includes a default entity model without specific conditions. The `dc:DefaultProfileSubset` class and is a subclass of the `dc:ProfileSubset` class.

The `dc:ConditionalProfileSubset` class represents a service-related conditional subset of an entity profile. A conditional profile subset includes an entity model that is valid when specific conditions exist. The `dc:ConditionalProfileSubset` class is a subclass of the `dc:ProfileSubset` class.

Class: `dc:Condition`

Condition – A condition that describes a specific situation

in-range-of: `dc:qualifiedBy`, `dc:consistsOf`

in-domain-of: `dc:consistsOf`

The `dc:Condition` class represents a condition. A condition describes a specific situation, e.g. the location or the activity of an entity, the time of day or the temperature within a room.

Class: `dc:EntityModel`

EntityModel – An entity model, e.g. a user model or a user group model

in-range-of: `dc:hasEntityModel`

in-domain-of:

The `dc:EntityModel` class represents a service-specific model of an entity, e.g. a user model, user group model or a model of another entity.

The `dc:UserModel` class represents a service-specific model of a user and is a subclass of the `dc:EntityModel` class.

The `dc:UserGroupModel` class represents a service-specific model of a user group and is a subclass of the `dc:EntityModel` class.

Class: dcs:Service

Service – A service

in-range-of: dcs:associatedWith

in-domain-of:

The `dcs:Service` class represents a service, e.g. a platform service or an end-user service.

Class: dcs:Context

Context – A context, e.g. a user location or time of day

in-range-of: dcs:consistsOf

in-domain-of: dcs:associatedWith

The `dcs:Context` class represents a context, e.g. a user location or time of day.

The `dcs:Activity` class represents an activity, and is a subclass of the `dcs:Context` class.

The `dcs:Location` class represents a location and is a subclass of the `dcs:Context` class.

The `dcs:Temperature` class represents a temperature and is a subclass of the `dcs:Context` class.

The `dcs:TimeOfDay` class represents a time of day and is a subclass of the `dcs:Context` class.

The `dcs:WeekDay` class represents a weekday and is a subclass of the `dcs:Context` class.

Property: dcs:hasName

Domain: dcs:ProfileSubset

Range: xsd:string

The `dcs:hasName` property indicates the name of a profile subset that is unique within a profile.

Property: dcs:hasDescription**Domain:** dcs:ProfileSubset**Range:** xsd:string

The `dcs:hasDescription` property indicates the description of a profile subset.

Property: dcs:hasOperator**Domain:** dcs:Condition**Range:** xsd:string

The `dcs:hasOperator` property indicates the operator of a relation, e.g. `equal`, `notEqual` or `greaterThan`.

5.4 Recommendations and Learning

This section depicts requirements, the current state of the art, and the SPICE WP4 ontology approach for the *Knowledge Interpretation Enabler*. In particular, this section deals with the specification of the *Knowledge Acquisition and Provisioning System (KAPS)* related recommender ontology

5.4.1 Requirements

The SPICE platform must be able to aggregate and to interpret information from multiple distributed *Knowledge Sources*. Moreover, this information must be used to infer additional knowledge in order to add “intelligence” to services. Knowledge refers to all information that can be used to provide contextual recommendations and/or proactive services. The resulting system must be highly dynamic, scalable, and loosely coupled. The *Knowledge Interpretation Enabler* (aka *KAPS*) provides this functionality. It infers knowledge by applying various learning and recommendations techniques.

The *KAPS* requires that all input data is tagged with some kind of timestamp so that, when a learning or recommendation process is triggered, the correct information (e.g. snapshot data, user models, or rules) could be gathered from the various, distributed knowledge bases and sources. Moreover, some parameters are “highly volatile”, i.e. the particular value of this parameter changes frequently. The usefulness of the gathered knowledge depends on the fact that the returned information must reflect the state at the exact point in time when the initial process (learning or recommending) was started.

This also imposes two major technical requirements on the *KMF*:

- A global time for synchronization of all *Knowledge Sources* is needed.
- *Knowledge Sources* have to provide past knowledge, too.

In addition to that, the *KAPS* requires that all learning and recommendation techniques, which have been deployed in the SPICE platform, could be re-used by a number of different services. This requirement is met by providing a common data model for all *KAPS* subsystems.

In the following, we provide an overview on the current state of the art and introduce our approach to a common data model for learning and recommendations.

5.4.2 State of the Art

Not much has been done in the field of ontology definitions for learning approaches and recommendation techniques. [Middleton2003Middleton2003] explained how the direct usage of domain-specific ontologies could help in better categorizing different contents that are to be recommended, and how this approach could lead to an overall improved quality of the provided recommendations. However, any user models that are learnt during the process have not been semantically defined and neither has the actual format of the provided recommendation been defined. The following paragraphs focus on the current state of the art in algorithms and user modelling in order to derive a feasible ontology model, which is then used in the learning process as well as in the recommendation process within the SPICE project in general and in the SPICE *KAPS* more specifically.

5.4.2.1 Learning

The observation-based learning falls into the field of data mining and machine learning. Some of existing techniques are presented as follows.

Decision tree learning

Decision tree describes a tree structure wherein leaves represent classifications and branches represent conjunctions of features that lead to those classifications. [Menzies2003] Every decision tree can be easily converted to a set of rules while the efficiency of rules heavily depends on the converting schema.

Artificial Neural Networks (ANN)

The model of ANN is represented as a function: $f: X \rightarrow Y$, where X can be a composition of another functions. ANN learning is powerful to infer a function while the dependency of facts is very complicated.

Association rule mining

The association rule learner tries to find the facts that often occur together while it doesn't care about the real meaning of the facts. This learning technique is widely used for marketing to infer the rules of customer purchase behaviour. The existing algorithms are AIS, SETM, APRIORI, Direct Hashing and Pruning and Dynamic Set Counting [Gobel2004].

Treatment learning

The treatment learner hunts for a minimal difference set between things instead of extensive descriptions and provides smaller models, because people often prefer simple models rather than intricate ones [Menzies2003].

In general, the learning result can be represented by rules directly (association rule mining, treatment learning) or be converted to rules (ANN, decision tree learning). A rule is a "If...Then..." statements. SWRL is a Semantic Web Rule Language which extends the OWL syntax of OWL to express rules. It introduces a new axiom *rule*. The basic syntax of *rule* is as follows.

```
rule ::= 'Implies(' [ URIreference ] { annotation } antecedent consequent ')'  
antecedent ::= 'Antecedent(' { atom } ')'  
consequent ::= 'Consequent(' { atom } ')'
```

It means when the antecedent is true, the consequent is also true.

In Jena, the rules are structured in a similar way. The rule comprises a list of antecedents (body) and a list of consequents (head) and is identified by a label (name). The body can be TriplePatterns (basic RDF statements) or Functors, and the head can be TriplePatterns, Functors or Rule.

5.4.2.2 Recommender

Today's recommender systems attempt to find web pages, products, or news that are relevant for us. Their recommendations are based on data about our past behaviour or statistical models. The resulting data, which is used to make recommendations, is stored in a user profile. Kobsa [Kobsa2004] and Stewart [Stewart2004] provide a good survey of user modeling techniques and De Roure [DeRoure2001] provides a review of recommender systems. The learning algorithms and prediction methods used by today's

recommender systems are strictly tailored to the service for which the system was designed. Moreover, these systems abstract from the prevailing situation during the learning process. This means that they could not provide reasonable recommendations, if the user employs the same service in different situations (such as 'being at home' or 'being at work') and acts differently.

Content-based recommender systems base their recommendations on the similarity between new items and items that a user has liked before (user-to-item). Examples of content-based recommender systems are Fab [Balabnovi1997] and ELFI [Schwab2000]. Fab recommends web pages and ELFI recommends funding information from a database.

Collaborative recommender systems derive their recommendations on the basis of content ratings from people with similar interests (either user-to-user or item-to-item). PHOAKS [Terveen1997] and Group Lens [Konstan1997] are examples of such recommender systems. PHOAKS recommends web links found in newsgroups. Whereas the Group Lens system recommends single newsgroup postings.

Calvin [Bauer2002] is a *personal information agent recommender* that monitors the user's browsing behaviour over time. It learns to identify broader contexts by relating documents that tend to be accessed together. Calvin extracts information about how the user tends to access groups of documents over time, and predicts document relevance based on similarity of the extended sessions within which various documents are accessed.

5.4.3 Approach

This section describes how specific knowledge like feedback, snapshots, items, rules and recommendations is represented in the *KAPS*.

5.4.3.1 Data Structures of the Learning Subsystem

The learning process in the *KAPS* works either on the basis of provided feedback or through observation. This section focuses on the necessary data structures for both kinds of learning input.

Feedback

All *feedback* is provided in terms of positive and negative individual ratings on service-specific items. Figure 36 shows the class diagram of the *Feedback* class. For the rating, an explicit rating scale (e.g. Likert scale) is used to provide a generic representation of the input data. A timestamp identifying the moment the feedback was given is also required.

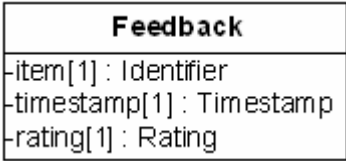


Figure 36: Feedback class

Snapshots

A *snapshot* represents a set of input variables, such as the contents of a shopping basket, which are monitored over time. In practice, these input variables are not continuously monitored. Samples are taken at specific points in time either at regular intervals, based on variable changes, or triggered by specific events. Table 3 shows a set of snapshots that can be used to learn golf-playing behaviour based on weather conditions (taken from [Menzies2003]). The learnt knowledge can be used to provide activity recommendations to golf players based on present weather conditions. All snapshot data, i.e. the actual values of the input variables, is retrieved from (predefined) *Knowledge Sources*.

Snapshot #	Outlook	Temp (°F)	Humidity	Windy	Golf-playing behaviour	Outlook = overcast	Timestamp
1	sunny	85	86	false	none	no	19/03/2006
2	sunny	80	90	true	none	no	08/04/2006
3	sunny	72	95	false	none	no	22/04/2006
4	rain	65	70	false	some	no	06/05/2006
...
13	overcast	72	90	true	lots	yes	09/09/2006
14	overcast	81	75	false	lots	yes	23/09/2006

Table 3: Log of weather conditions and golf-playing behaviour (example)

LearntRule

The knowledge acquired by observation-based learner is represented as rules. The rule is an “If ... then ...” statement, describing the association of two sets of facts. The structure of *LearntRule* consists of the following members.

- *ID* is a unique identifier of each rule.
- *Supp (optional)* represents how often the condition and consequence item sets co-occur in a snapshot.
- *Conf (optional)* represents how often the consequence follows the condition in a snapshot.
- *Scope* presents the learning scope of the rule, i.e. the scope of the consequence in the rule. For example, if the scope is “modality”, the consequence is the fact describing audio, video or text.
- *Condition* is a set of facts.
- *Consequence* is a set of facts.

The *support* and *condition* members are relevance parameters for association rules. Theoretically, *condition* and *consequence* can contain multiple facts. However the more facts in either side the more complicated the rule is. Study on the human decision making finds out that people are not interested in complicated models. People often use simple models to make decision, only considering the key information that can lead to the most benefits rather than all information they have [Menzies2003] [Gigerenzer1996]. Based on this concern, in order to acquire the most useful knowledge, condition and consequence are limited to proper sizes in learning processes for different services.

LearntRuleSet

A LearntRuleSet holds of all rules retrieved in a learning process. It consists of a list of LearntRules, a learning scope, the minimum support and confidence thresholds for all rules in the list. Figure 37 depicts the data structure of *LearntRuleSet* and *LearntRule* classes.

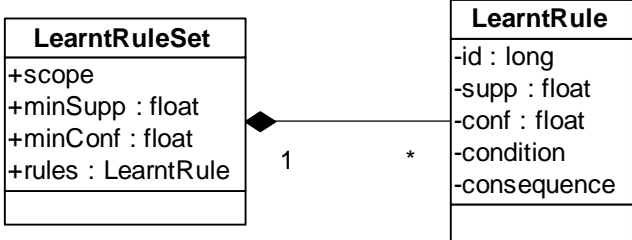


Figure 37: *LearntRuleSet* and *LearntRule* classes

5.4.3.2 Data Structures of the Recommender Subsystem

The components of the recommender subsystem provide recommendations that consist of items that are relevant to either real-world entities (users, groups) or computational entities (services) in given situations. Depending on the type of the requested recommendation, an item describes a particular modality (audio only, video, text), content (music, news, movies), service type (music store, news portal, cinema ticket reservation system), or action. This section focuses on the necessary data structures for items and recommendations.

Items and Features

An *item* abstracts from the actual object of the recommendation while providing a computer-readable (semantic) description. This description can be used to improve the learning process as well as the recommendation delivery process. Figure 38 shows the class diagram of the `Item` class. This class consists of a set of attributes, such as a unique identifier (“id”), a friendly¹ name (“name”), a type (“type”), and an optional list of features (“features”).

¹ The term “friendly” means human-readable in this case.

A *feature* provides additional information about an item, such as applicable categories or tags. The Feature class consists of a list of attributes, such as a unique identifier (“id”), a friendly name (“name”), and an optional weight (“weight”).

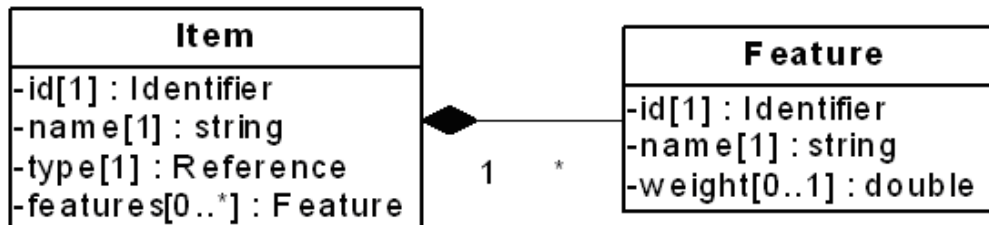


Figure 38: *Item* and *Feature* classes

Recommendations

A *recommendation* consists of all items that are relevant to either real-world entities (users, groups) or computational entities (services) in a given situation. Figure 39 shows the class diagram of the corresponding `Recommendation` class. This class consists of a set of attributes, such as a type (“type”), a list of recommended items (“items”), the mean relevance of all recommended items (“mean_score”), the mean confidence in the recommendation (“mean_confidence”), and the mean support for the recommendation (“mean_support”). The last three attributes are optional and depend on the applied algorithm.

The *list of recommended items* is a set of object instances of the `RecommendedItem` class. This class consists of a set of attributes, such as a reference to the recommended item (“item”), the exact relevance of this item (“score”), the confidence in the recommendation (“confidence”), and the support for the recommendation (“support”). The last three attributes are optional and depend on the applied algorithm.

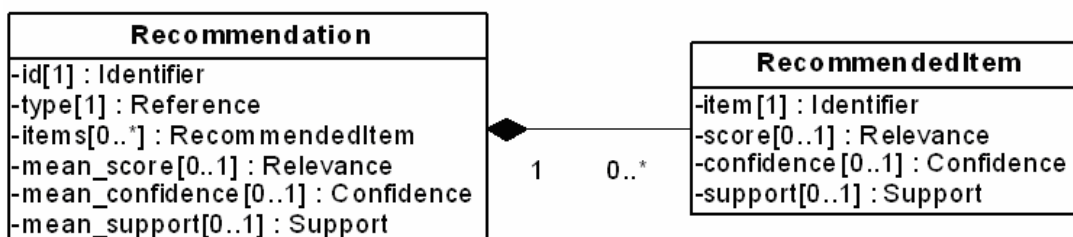


Figure 39: *Recommendation* and *RecommendedItem* classes

The data structures of the learning and recommender subsystem that is depicted in Figure 37, Figure 38 and Figure 39 now have to be transformed into an ontology definition. As ontology language we chose OWL. A visualisation of the resulting ontology definition is depicted in Figure 40 and Figure 41. The detailed class and property definitions are described in section 5.4.5.

5.4.4 Visualisation of Recommendation and Learning Ontology

Figure 40 visualises the recommender subsystem related concepts and relationships.

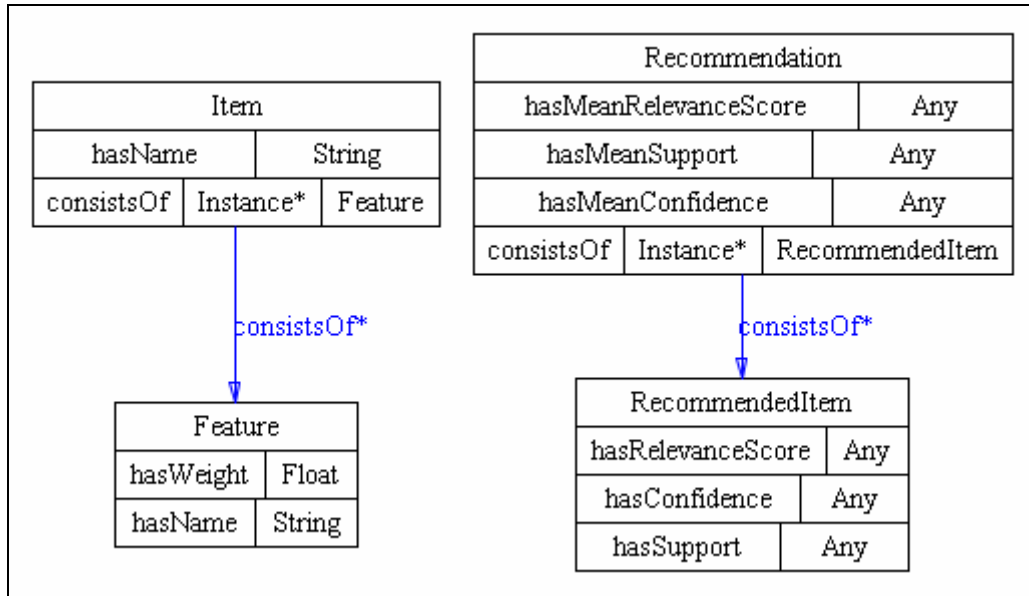


Figure 40: Recommender subsystem and related concepts visualisation

Figure 41 visualises the ontology definition of the learning subsystem, in particular `LearntRuleSet` and `LearntRule`.

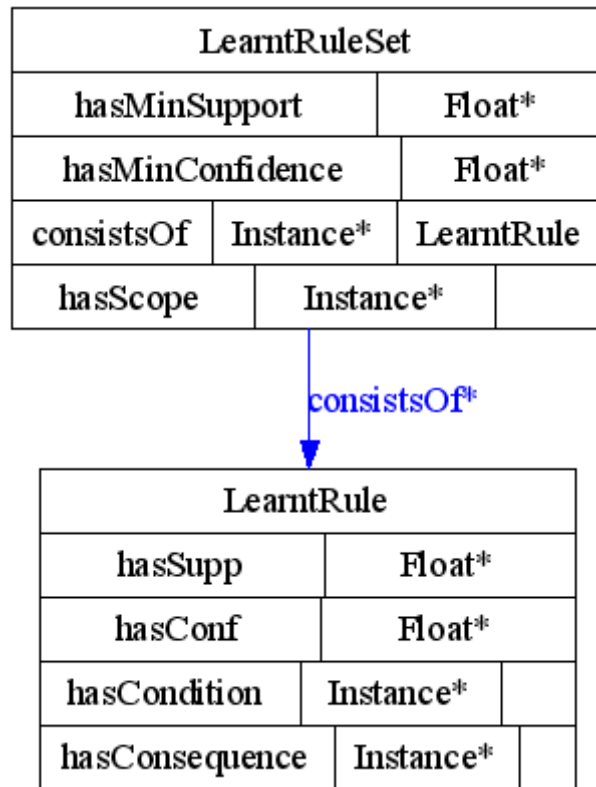


Figure 41: Learning ontology visualisation

Figure 42 illustrates the possible instances in a snapshot. It includes the instances of time (“evening”), location (living room), device (“IBM laptop”), network (BT broadband) and modality (“audio” and “video”).

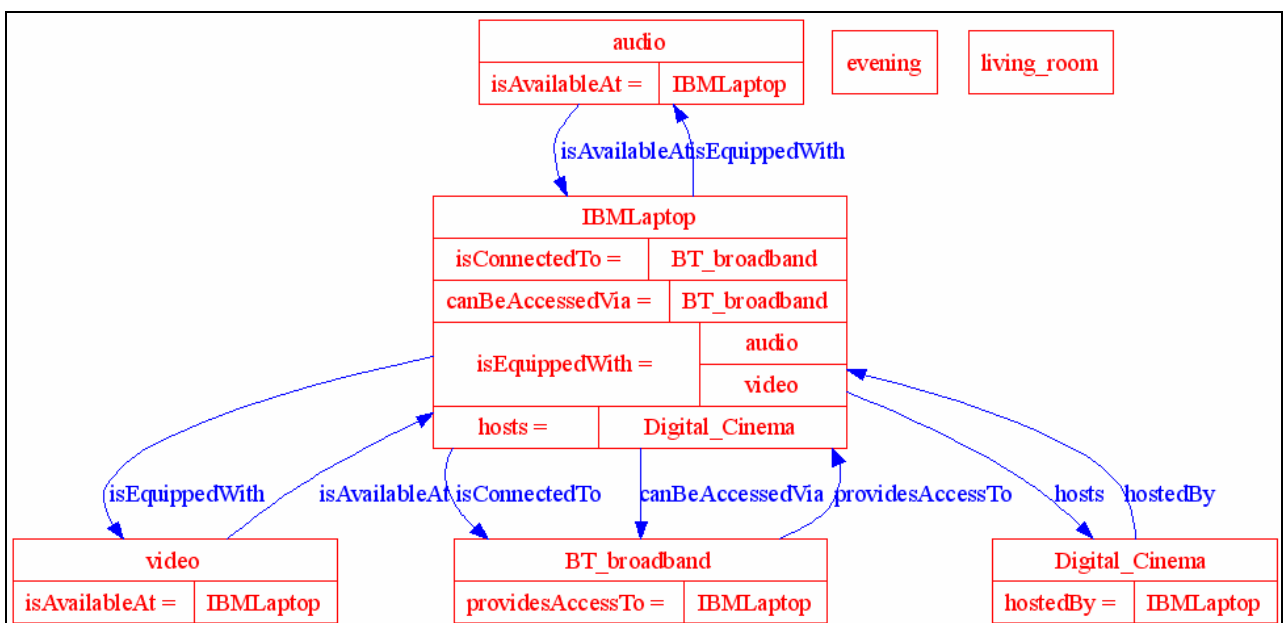


Figure 42: Snapshot visualisation

5.4.5 Recommendation and Learning Ontology Definition

Learning:

Class dcs: LearntRuleSet

RuleSet – A collection of association rules.

in-range-of: dcs:associatedWith

in-domain-of: dcs:consistsOf

The dcs: LearntRuleSet class consists of all association rules retrieved in a specified learning scope. This class has a set of attributes, a learningScope, a set of rules, the minimum support and the minimum confidence of all rules. For example, a set of modality learning rules for a multimedia service with minimum support 70% and minimum confidence 60%.

Class dcs: LearntRule

LearntRule – an association rule.

in-range-of: dcs:associatedWith

in-domain-of: dcs:consistsOf

The dcs: LearntRule class represents rules retrieved from the learning process. It consists of a set of attributes, such as an id, the particular support and confidence parameters, and the condition and consequence of the rule

Property: dcs:hasMinSupp

hasMinSupport – minimum supp.

Domain: dcs: LearntRuleSet

Range: xsd:float

The dcs:hasMinSupp property indicates that all rules in the rule set have a threshold the minimum support.

Property: dcs:hasMinConf

hasMinConfidence – The minimum conf.

Domain: dcs: LearntRuleSet

Range: xsd: float

The `dcs:hasMinConf` property indicates that all rules in the rule set have a threshold the minimum confidence.

Property: dcs:hasSupp

hasSupp – The support of a rule.

Domain: dcs: LearntRule

Range: xsd: float

The `dcs:hasSupp` property indicates the support of the rule.

Property: dcs:hasConf

hasConf – The confidence of a rule.

Domain: dcs: LearntRule

Range: xsd: float

The `dcs:hasConf` property indicates the confidence of the rule.

Property: dcs:hasCondition

hasCondition – The condition part of a rule.

Domain: dcs: LearntRule

Range:

The `dcs:hasCondition` property indicates the condition part of the rule.

Property: dcs:hasConsequence

hasConsequence – The consequence part of a rule.

Domain: dcs: LearntRule

Range:

The `dcs:hasConsequence` property indicates the consequence part of the rule.

Recommendations:**Class: dcs:Recommendation**

Recommendation – A recommendation, e.g. in the context of the recommender engine.

in-range-of: dcs:associatedWith

in-domain-of: dcs:consistsOf

The `dcs:Recommendation` class consists of all items that are relevant to either real-world entities (users, groups) or computational entities (services) in a given situation. This class consists of a set of attributes, such as a type, a list of recommended items, the mean relevance of all recommended items, the mean confidence in the recommendation, and the mean support for the recommendation. The last three attributes are optional and depend on the applied algorithm.

Class: dcs:RecommendedItem

RecommendedItem – a recommended item.

in-range-of: dcs:consistsOf

in-domain-of: dcs:hasRelevanceScore; dcs:hasSupport; dcs:hasConfidence

The `dcs:RecommendedItem` class abstracts from the actual object of the recommendation while providing a semantic description. This description can be used to improve the learning process as well as the recommendation delivery process. This class consists of a set of attributes, such as the exact relevance of this item, the overall confidence in the recommendation, and the overall support for the recommendation.

Class: dcs:Feature

Feature – a feature.

in-range-of: dcs: associatedWith

in-domain-of: dcs:hasWeight

The `dcs:Feature` class provides additional information about an item, such as related categories or tags. The Feature class consists of a list of attributes, such as a unique identifier, a friendly/human-readable name, and an optional weight.

Property: dcs:hasMeanRelevanceScore

hasMeanRelevanceScore – mean relevance score.

Domain: dcs: Recommendation

Range: xsd:float

The `dcs:hasRelevanceScore` property relates a particular recommendation to the mean relevance score for the recommendation.

Property: dcs:hasMeanSupport

hasMeanSupport – mean support.

Domain: dcs: Recommendation

Range: xsd:float

The `dcs:hasMeanSupport` property relates a recommendation to the mean support for the recommendation.

Property: dcs:hasMeanConfidence

hasMeanConfidence – The mean confidence.

Domain: dcs: Recommendation

Range: xsd:float

The `dcs:hasMeanConfidence` property relates a particular recommendation to the mean confidence for the recommendation.

Property: dcs:hasRelevanceScore

hasRelevanceScore – The relevance score.

Domain: dcs: RecommendedItem

Range: xsd:float

The `dcs:hasRelevanceScore` property relates a recommended item to its relevance score.

Property: dcs:hasSupport

hasSupport – The support.

Domain: dcs: RecommendedItem

Range: xsd:float

The `dcs:hasSupport` property relates a recommended item to its support.

Property: dcs:hasConfidence

hasConfidence – The confidence.

Domain: dcs: RecommendedItem

Range: xsd:float

The `dcs:hasConfidence` property relates a recommended item to its confidence.

Property: dcs:hasWeight

hasWeigh – The weight.

Domain: dcs: Feature

Range: xsd:float

The `dcs:hasWeight` property relates a feature to its weight.

5.5 Presence

In this section we describe the requirements and definition of the SPICE Rich Presence Ontology. The Presence ontology will be used as part of the RDF-PIDF mapping component in the IMS Gateway.

5.5.1 Requirements

The inter-working between RDF and PIDF worlds has some impacts on the Rich Presence-related ontology in the *KMF* world. In general, a variety of PIDF-based standards cover a large amount of knowledge concepts related to presence and location of entities. Defining an inter-working function hence does require the concepts addressed by PIDF on the one hand, and by the SPICE Rich Presence ontology on the other hand to be compatible and quite close to ease as much as possible the conversion process.

The following gives the main steps of the inter-working operation between RDF and PIDF where ontology is involved.

- If an RDF SPICE component wants to deal with the IMS Presence Server, it has first to register the user using an RDF structure parameter containing a unique User identifier `userID@domainID` where `userID` can be a username, telephone-number (like `Teresa@imsasg.com` and `0664421246@imsasg.com`).
- To change the presence status of a user in the IMS subsystem, the RDF SPICE component sends to the IMS-GW the RDF data containing the user identification and the attributes to be changed. For example:

```
<Person rdf:ID="Teresa@imsasg.com">
  <hasLocation>
    <Location rdf:ID="Room1"/>
  </hasLocation>
</Person>
```

- The RDF2PIDF mapping component is called to translate this RDF document to a PIDF document according to the mapping between the PIDF standards and the Rich Presence ontology. The SIP URI (SIP identifier of the person) is retrieved by the IMS-GW from the Person identifier; a PIDF Presence document is built and then published on behalf of that user to the IMS Presence server (acting as Presence Network Agent according to 3GPP specifications).

Conversely, when a new PIDF document is notified to the IMS-GW by the IMS Presence Server, this document is translated in an RDF document using the RDF2PIDF mapping component and referring to the SPICE Rich Presence ontology defined below. Then, IMS-GW notifies all RDF SPICE knowledge consumer components that have subscribed to the presence status modification for the retrieved user.

5.5.2 State of the Art

OMA has been working on a set of enablers dealing with, amongst others:

- Presence information (Presence)
- Device characteristics (UAPProf)
- Resource lists (XML Document Management (XDM) – Not addressed in this document)
- Presence authorisation rules (XDM)

In some cases (Presence and XDM), the data models used are relying on IETF specifications.

The information and concepts provided by these standards are illustrated below. Such standards are essential to be captured as is from the “conceptual” perspective as they are natively supported by IMS/OMA-based mobile terminals.

5.5.2.1 Presence Data

OMA has been standardizing the SIMPLE Presence service enabler [Presence] that reuses different data models defined by IETF for representing “presence” information, as well as defining its own extensions.

IETF has defined the Basic Presence Data model (RFC3863), Generic Presence Data Model (RFC4479) and Rich Presence Information (RFC4480).

Basic Presence Data model (RFC3863)

PIDF (Presence Information Data Format) [RFC3863] is the very minimal data format for representing Presence information in IETF. It consists of a set of tuples that can have an open or closed status. These tuples define whether one is available to communicate through a specific service (open) or not (closed).

Presence Data Model (RFC4479)

The Generic Presence Data Model (RFC4479) [RFC4479] is enriching the basic model by making distinction between the kinds of tuples. It defines ‘services’, ‘persons’ and ‘devices’. This enables it to connect persons to services and to show which kind of devices the person uses in the particular service. It also potentially allows multiple persons (groups) to be represented under the same presentity.

Rich Presence Information (RFC4480)

A third model is the Rich Presence Information (RFC4480) [RFC4480]. This model takes also in account the user activity, user mood and some description of place the user is in. Most of these values (for moods, activities) are standardized and refer to other RFCs. Such standardized values should be considered as the basics in the SPICE ontology as it will ease very much conversion to and from standards into semantic representations for reasoning.

OMA Presence Specific Information

[Presence] defines its own additional information to the presence for OMA devices. Interesting concepts like willingness to communicate, service/application description (version, name, etc), access network type are added that can be relevant for SPICE and integration with services.

5.5.2.2 Location Information

IETF RFC4119

This vocabulary, standardized by RFC4119 [RFC4119] and reused in OMA Presence Service [Presence], defines a format for location-based information both for geolocation (GPS coordinates) and for civil location. It can also identify the localization method, the source of the information and the usage rules. The part of the usage rules is interesting because this covers a part of privacy, which is essential in filtering the information to be provided by the IMS-GW on the IMS side, rather than for setting up the privacy level of original PIDF information on the *KMF* side.

IETF RFC4589

RFC4589 [RFC4589] is a vocabulary which standardizes the types of places where a user or device can be found. For example café, train, aircraft, etc but also indoor/ outdoor and public. It defines different physical spaces by their name from which a lot of information can be derived in the sense of context awareness and knowledge in general.

5.5.3 Approach

Based on the state-of-the-art, and on the inter-working function to be performed between the *KMF* and the IMS to accommodate native IMS *Knowledge Sources* (Presence User Agent) and consumers (Presence Watcher), the SPICE Rich Presence ontology needs to be built very closely on IMS/OMA-based standards, at least regarding the concepts it addresses. Being close to standardized solutions has some advantages, for example, by making it easier to reuse models or parts of models in other solutions created by other parties, and being able to convert concepts expressed in “native” standard format to and from a semantic representation (for example to extract semantic knowledge from a native presence source or on the other way to publish some presence information based on a semantically-inferred knowledge). “Scanning” the standardized solutions as above also prevents from reinventing the wheel, as well as facilitates the identification of new needs (new information) not yet captured by standards that could be proposed in the future as extensions.

The SPICE Rich Presence ontology captures the concepts and properties addressed in the IMS/OMA standard models as this is an important base for representing context information for persons/groups, devices and services in SPICE.

The ontology described in this section is built from these standards with the goal to be easily extended with other SPICE- relevant information as described in the previous sections of this deliverable. The ontology includes all PIDF concepts covered by the IMS/OMA standards.

5.5.4 Visualisation of Presence Ontology

Figure 43 shows how the Presence Data Model described in RFC 4479 is represented in the Rich Presence Ontology.

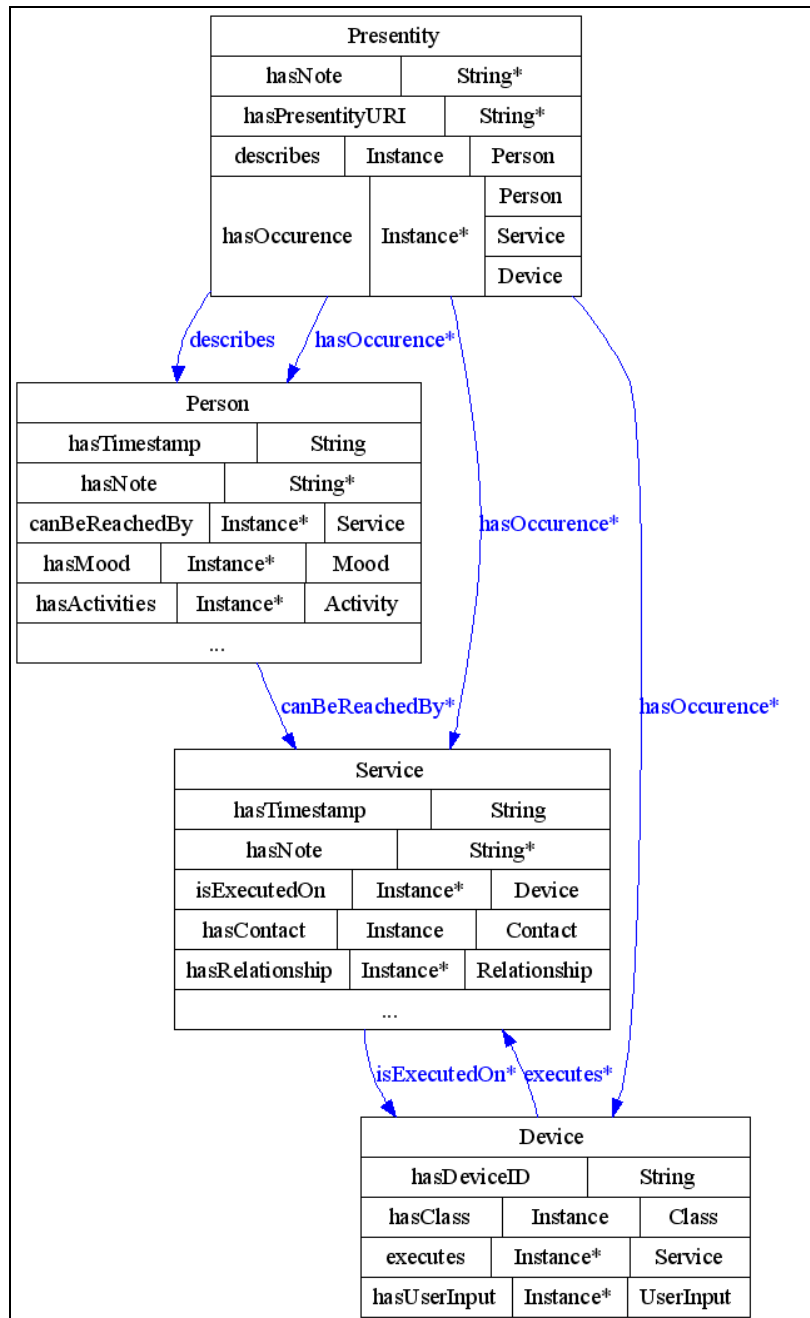


Figure 43: Presentity and related concepts

Figure 44 and Figure 45 show the part of the ontology of concepts related to a Person that is represented by the Presentity. In particular this includes basic presence information defined in RFC 3863 and rich presence information as defined in RFC 4480.

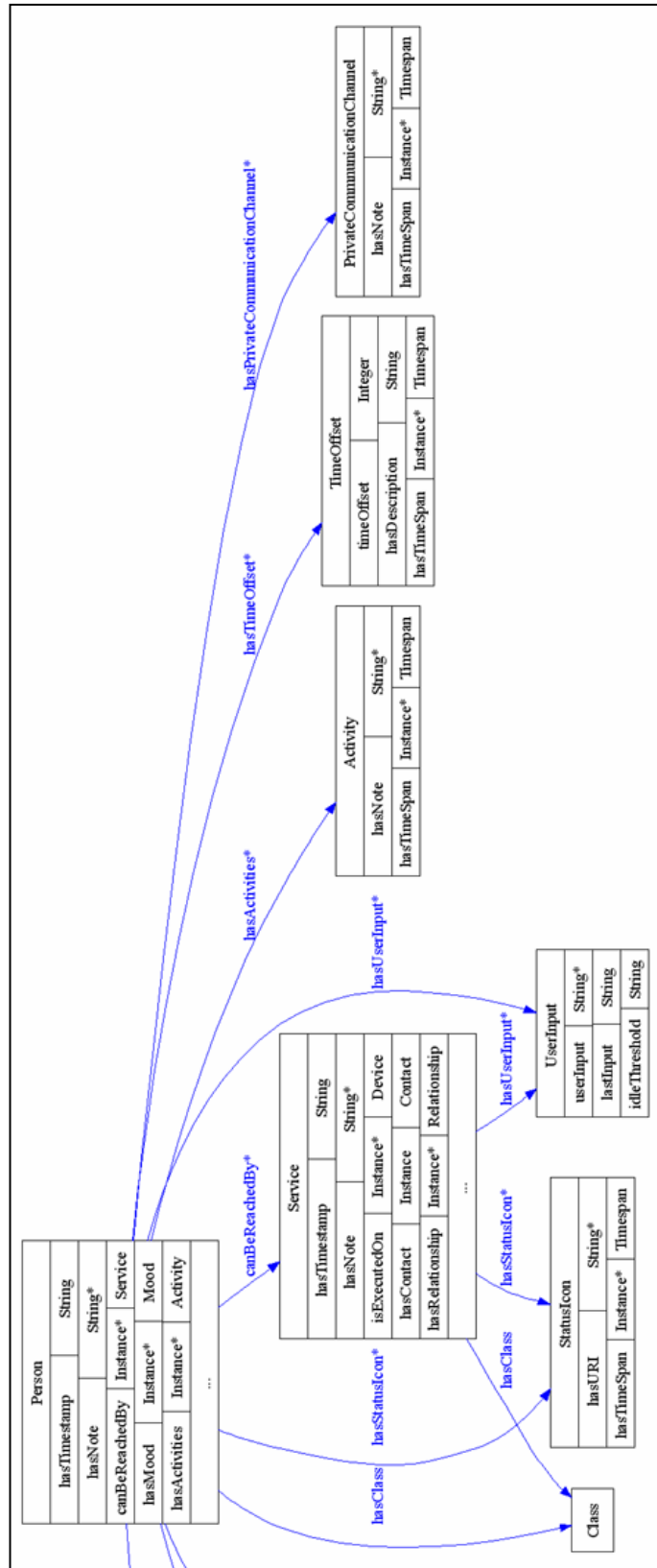


Figure 44: Person and related concepts (part 1)

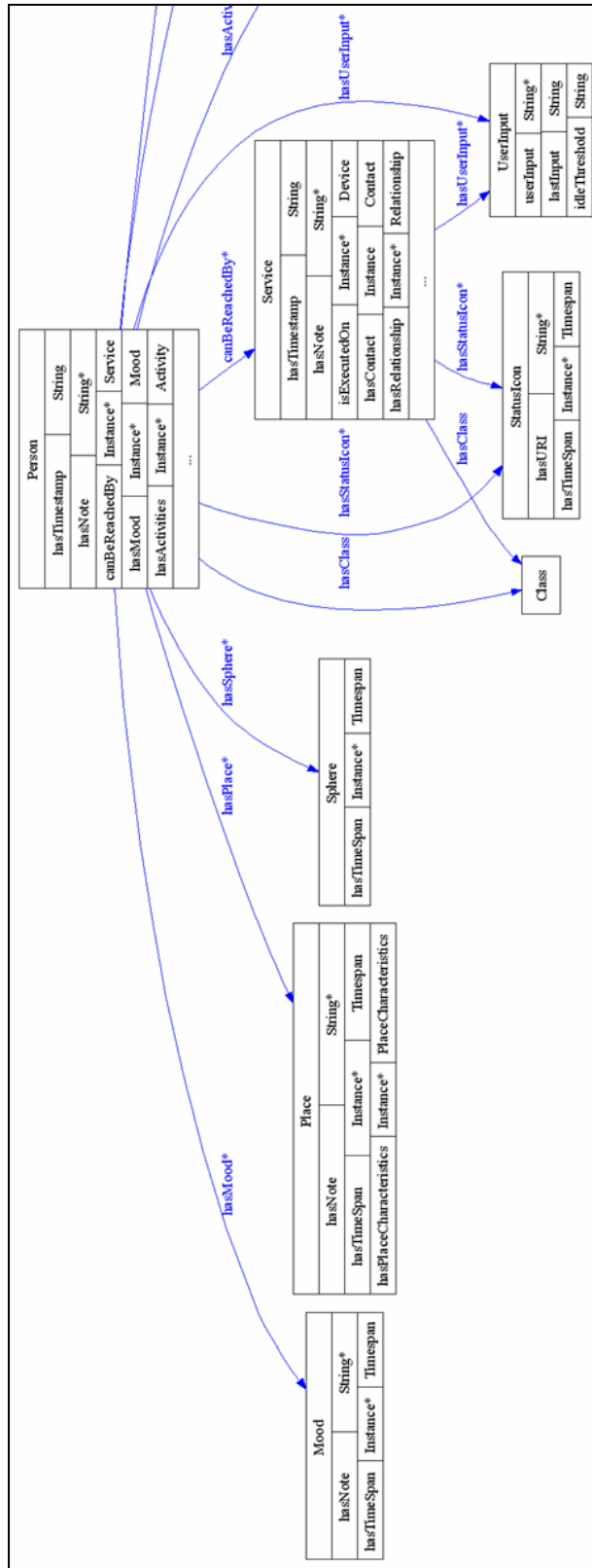


Figure 45: Person and related concepts (part 2)

Figure 46 shows how a service is describe in our ontology.

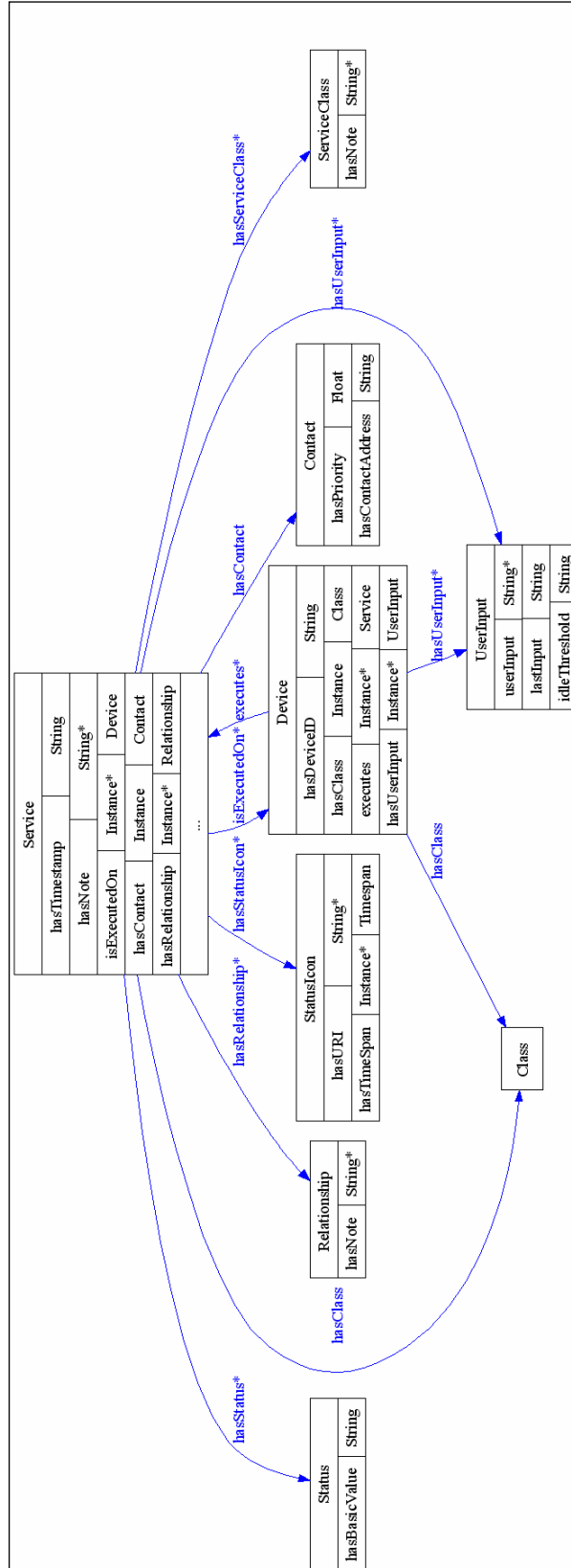


Figure 46: Service and related concepts

Figure 47 shows how a device is represented in the SPICE rich presence ontology.

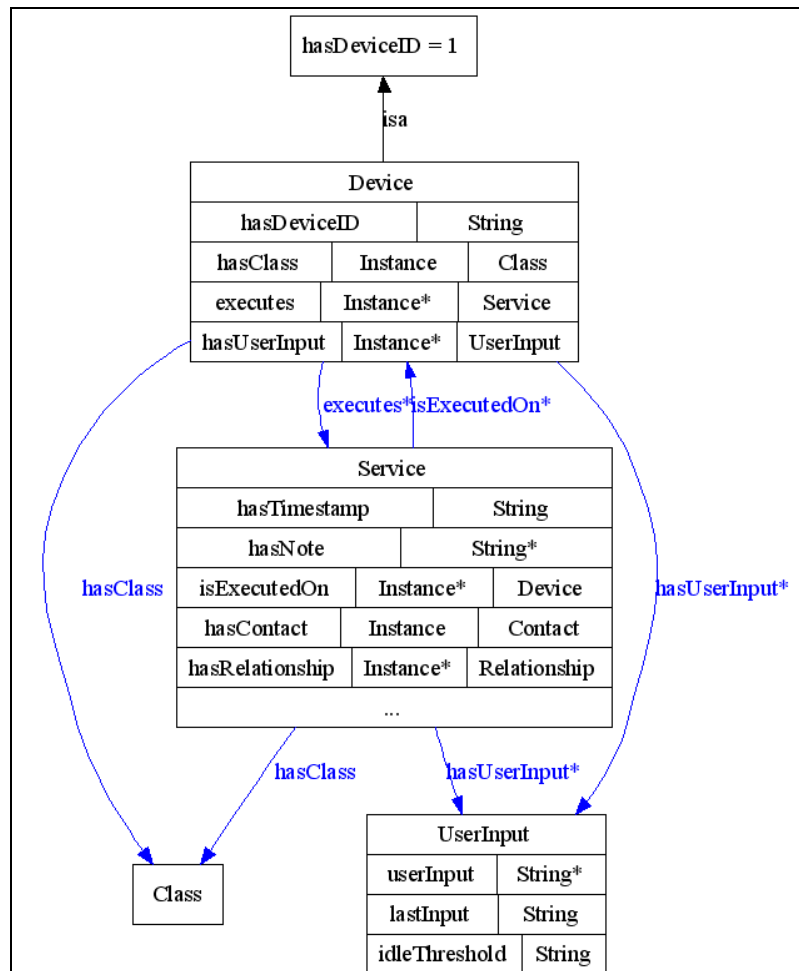


Figure 47: Device and related concepts

Figure 48 shows how a place is described. As an example we show the Street concept that we represent as a subclass of Place. In a later version we will add other place types compliant with RFC 4480.

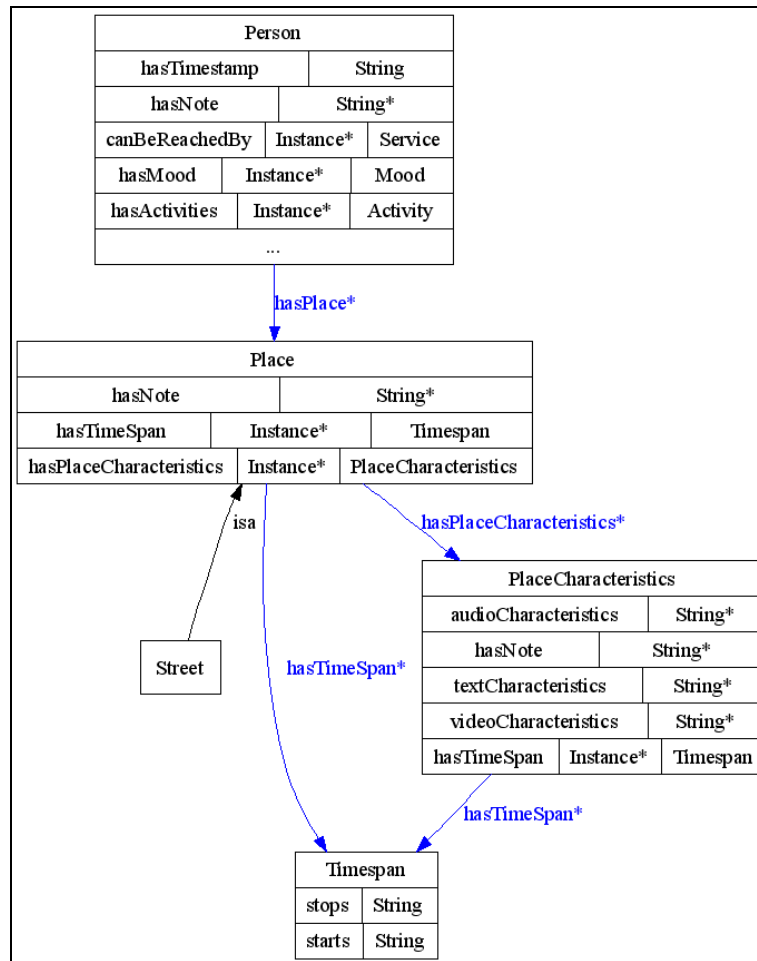


Figure 48: Place and related concepts

5.5.5 Presence Ontology Definition

Since the definition of the presence ontology is very comprehensive, the documentation for all classes and properties can be found at [PresenceOntology].

6 Conclusion

In this document we presented an overview of several ontologies for the use and exchange of information that is related to SPICE WP4 enablers. In particular, we presented physical space, service context, recommendations and learning, user profile and presence ontologies. In addition, we included a discussion and investigation on different approaches to modelling relationship attributes for the representation of uncertainty and Quality of Context (QoC).

In order to enable a comprehensive understanding of the presented ontologies, we depicted their relation to the WP4 architecture and the overall SPICE project. In particular, section 2 depicts the relation of WP4 ontologies to the overall SPICE reference model and to the Distributed Communication Sphere (DCS) of WP3. In section 3, we briefly introduce the different enabler groups of the WP4 architecture. For a more detailed description of the WP4 architecture, see [SPICE D4.4]. Section 3 also includes an elaboration on which enabler group deals with which type of information, i.e. with which type of ontology, and on how these ontologies are used within these enabler groups.

Subsequently, section 4 presents a detailed investigation on different approaches to modelling relationship attributes for the representation of uncertainty and Quality of Context (QoC) as a solution that can be used throughout the complete WP4 architecture. Finally, section 5 includes the actual ontology definitions together with the enabler specific requirements, state-of-the-art and visualisation of the ontology approaches. Since the presented ontologies are subject to further refinement and alignment, the ontology definition should not be considered final in the current status.

The resulting ontology definitions will enable the inter-working between different WP4 enabler groups, and between WP4 enablers and various other SPICE platform components. In the end, this semantic-based inter-working will enable the provision of meaningful and personalized service access to end-users.

7 References

Bookmark	Description
[Balabnovi1997]	M. Balabanovi, Y. Shoham, “Fab: Content-based, collaborative recommendation”, Communications of the ACM, volume 40, no. 3, 1997.
[Bauer2002]	T. Bauer, D. Leake, “Exploiting information access patterns for context-based retrieval”, Proceedings of the 7 th international conference on Intelligent user interfaces, San Francisco, California, USA, pages 176-177, ACM Press, 2002.
[Brussee2005]	Rogier Brussee, Stanislav Pokraev, “Reasoning on the Semantic Web,” Chapter in Semantic Web Services: Theory, Tools and Applications, Cardano, J. ed. (2006) to appear.
[CCPP]	W3C: Composite Capabilities/Preference Profiles (CC/PP): Structure and Vocabularies 1.0, 15 January 2004. Available: http://www.w3.org/Mobile/CCPP/ , 9 Nov 2006.
[daCosta2005]	da Costa, P.C.G.; Laskey, K.B. & Laskey, K.J. “PR-OWL: A Bayesian Ontology Language for the Semantic Web”, ISWC-URSW, 2005, 23-33.
[DeRoure2001]	D. De Roure, W. Hall, S. Reich, G. Hill, A. Pikrakis, M. Stairmand, “MEMOIR – an open framework for enhanced navigation of distributed information”, Information Processing and Management, issue 37, pp. 53 – 74, 2001.
[Ding2004]	Ding, Z. & Peng, Y. “A Probabilistic Extension to Ontology Language OWL”, HICSS '04: Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4, IEEE Computer Society, 2004.
[Euzenat2004]	J. Euzenat. An API for ontology alignment. In <i>Proceedings of ISWC</i> , pages 698–712, 2004.
[FOAF]	FOAF Vocabulary Specification, July 2005. Available: http://xmlns.com/foaf/0.1/ , 9 Nov 2006.
[Fukushige2005]	Fukushige, Y. “Representing Probabilistic Relations in RDF”, ISWC-URSW, 2005, 106-107.
[FuzzyDL]	FuzzyDL. Available: http://gaia.isti.cnr.it/~straccia/ , 9 Nov 2006.

- [Gigerenzer1996] G. Gigerenzer and D.G. Goldstein, "Reasoning the Fast and Frugal Way: Models of Bounded Rationality," *Psychological Rev.*, vol. 103, 1996, pp. 650-669.
- [Gobel2004] Max Gobel, *Data Mining for Self-Adapting Smart Home Environment*, Diploma thesis, Technical University Berlin, 2004.
- [GUP] 3GPP Generic User Profile (GUP), "Data Description Method (DDM)," 3GPP TR 23 941 v6.0.0, Dec 2004. Available: <http://www.3gpp.org/>, 9 Nov 2006.
- [Jena] Jena – A Semantic Web Framework for Java. Available: <http://jena.sourceforge.net/index.html>, 9 Nov 2006.
- [Kobsa2004] A. Kobsa, "Generic User Modeling System", *User Modeling and User-Adapted Interaction*, issue 11, pp. 49 – 63, Kluwer Academic Publishers, 2001.
- [Konstan1997] Konstan, J.A. Miller, B.N. Maltz, D. Herlocker, J.L. Gordon, L.R. Riedl, J. "GroupLens: applying collaborative filtering to Usenet news", *Communications of the ACM* Volume 40, No. 3, 1997.
- [Menzies2003] T. Menzies, Y. Hu, *Data Mining For Very Busy People*. *IEEE Computer*, October 2003, pgs. 18-25.
- [Middleton2003] Middleton, S. E., Shadbolt, N. R. and De Roure, D. C. (2003), "Ontology-based Recommender Systems", in Staab, S. and Studer, R., Eds. *Handbook on Ontologies in Information Systems*. Springer, 2003.
- [N-ary Relations] W3C: *Defining N-ary Relations on the Semantic Web*, April 2006. Available: <http://www.w3.org/TR/swbp-n-aryRelations/>, 9 Nov 2006.
- [Orwant1995] J. Orwant. *Heterogeneous Learning in the Doppelgänger User Modeling System*. *User Modeling and User-Adapted Interaction*, 4:107–130, 1995.
- [OTA] OpenTravel Alliance, 2001, Available: <http://www.opentravel.org>, 9 Nov 2006.
- [OWL] W3C: *OWL Web Ontology Language Reference*, 10 February 2004. Available: <http://www.w3.org/TR/owl-ref/>, 9 Nov 2006.
- [OWL-API] OWL-API. Available: <http://owl.man.ac.uk/api.shtml>, 9 Nov 2006.

- [OWL-S] OWL-S. Available: <http://www.daml.org/services/owl-s/1.0/>, 9 Nov 2006.
- [Pool2005] Pool, M.; Fung, F.; Cannon, S. & Aikin, J, "Is It Worth a Hoot? Qualms about OWL for Uncertainty Reasoning", ISWC-URSW, 2005, 1-11.
- [Presence] OMA Presence Simple specification, Available: http://www.openmobilealliance.org/release_program/Presence_simple_v1_0.html, 9 Nov 2006.
- [PresenceOntology] SPICE Presence Ontology. http://www.ist-spice.org/mobile_ontology/standards/presence/doc/index.html, to be published.
- [RDF] W3C: Resource Description Framework (RDF): Concepts and Abstract Syntax, Available: <http://www.w3.org/TR/rdf-concepts>, 9 Nov 2006.
- [RDF-Primer] W3C: RDF Primer, Available: <http://www.w3.org/TR/rdf-primer/>, 9 Nov 2006.
- [RDFS] W3C: RDF Vocabulary Description Language 1.0: RDF Schema, 10 Feb. 2004. Available: <http://www.w3.org/TR/rdf-schema>, 9 Nov 2006.
- [RFC3863] RFC3863, Available: <http://www.ietf.org/rfc/rfc3863.txt>, 9 Nov 2006.
- [RFC4119] RFC4119, Available: <http://www.ietf.org/rfc/rfc4119.txt>, 9 Nov 2006.
- [RFC4479] RFC4479, Available: <http://www.rfc-editor.org/rfc/rfc4479.txt>, 9 Nov 2006.
- [RFC4480] RFC4480, Available: <http://www.rfc-editor.org/rfc/rfc4480.txt>, 9 Nov 2006.
- [RFC4589] RFC4589, Available: <http://www.ietf.org/rfc/rfc4589.txt>, 9 Nov 2006.
- [Roman2002] Román, M.; Hess, C.K.; Cerqueira, R.; Ranganathan, A.; Campbell, R.H. & Nahrstedt, K. Gaia: A Middleware Infrastructure to Enable Active Spaces IEEE Pervasive Computing, 2002, 2, 74-83.
- [Schwab2000] I. Schwab, W. Pohl, I. Koychev, "Learning to Recommend from Positive Evidence", Proceedings of Intelligent User

Interfaces, pp. 241 – 247, ACM Press, 2000.

- [SPARQL] W3C: SPARQL Query Language for RDF, W3C Working Draft 4 October 2006. Available: <http://www.w3.org/TR/rdf-sparql-query/>, 9 Nov 2006.
- [SPICE D1.3] W. Goix et al. “Initial Architecture Design”, EU IST SPICE project deliverable D1.3, November 2006.
- [SPICE D3.1] Zhdanova, A.V., Boussard, M., Cesar, P., Clavier, E., Gessler, S., Hesselman, C., Kernchen, R., Le Berre, O., Marengo, M., Melpignano, D., Nani, R., Patrini, L., Strohbach, M., Villalonga, C., Vitale, A. “Ontology Definition for the DCS and DCS Resource Description, User Rules”, EU IST SPICE project deliverable D3.1, October 2006.
- [SPICE D4.4] Miquel Martin et. al., “Service Enablers Demonstration (Mock-ups Highlighting Service Intelligence in an Outdoor Extended LBS Scenario)”, EU IST SPICE project deliverable D4.4, October 2006.
- [Steward2004] A. Stewart, C. Niederée and B. Mehta, “State of the Art in User Modelling for Personalization in Content, Service and Interaction”, NSF/DELOS Report on Personalization, 2004.
- [Straccia2005] U. Straccia, “A fuzzy description logic for the semantic web”, Capturing Intelligence, 2005, 1.
- [Suryanarayan2002] Lalitha Suryanarayana, Johan Hjelm, “Profiles for the Situated Web”, International World Wide Web Conference, Honolulu, Hawaii, USA, 2002.
- [Terveen1997] L. Terveen, W. Hill, B. Amento, D. McDonald, J. Creter, “PHOAKS: a system for sharing recommendations”, Communications of the ACM, volume 40, no. 3, 1997.
- [UAProf] OMA: User Agent Profile (UAProf) v2.0, May 2003. Available: <http://www.openmobilealliance.org/>, 9 Nov 2006.
- [vanKranenburg2005] H. Van Kranenburg and H. Eertink, “Processing Heterogeneous Context Information”, Proceedings of 2005 Symposium on Applications and the Internet, Next Generation IP-based Service Platforms for Future Mobile Systems workshop, (SAINT 2005), ISBN 0-7695-2263-7, pp. 140-143 2005.
- [vanKranenburg2006] H. Van Kranenburg, M.S. Bargh, S. Iacob, and A. Peddemors, “A Context Management Framework for Supporting Context Aware Distributed Applications”, IEEE Com Mag, August



2006, vol. 44, nr. 8, pp. 67-74, 2006.

[XML]

W3C: Extensible Markup Language (XML) 1.0 (Third Edition),
04 Feb. 2004. Available: <http://www.w3.org/TR/REC-xml/>, 9
Nov 2006.

[Zimmer2006]

Zimmer, T. QoC: Quality of Context - Improving the
Performance of Context-Aware Applications Advances in
Pervasive Computing. Adjunct Proceedings of Pervasive
2006.