Grant Agreement N° 215483

Title:	*Codified Human-Computer Interaction (HCI) Knowledge and Context Factors*
Authors	*City, UniDue*
Editor:	*Neil Maiden (City)*
Reviewers:	*Christos Nikolaou (University of Crete)*
	Elisabetta Di Nitto (Politecnico di Milano)
Identifier:	*Deliverable PO JRA 1.1.3*
Type:	*Deliverable*
Version:	*6*
Date:	*1st_March_2009*
Status:	*Final*
Class:	*External*

Management Summary

This deliverable reports the results from preliminary, exploratory research to explore the potential impact of different types of codified HCI knowledge and context factors on the development, deployment and adaption of service-based software applications. It reports additional literature review results undertaken to inform the research, scopes and structures the preliminary research through presentation of a series of conceptual meta-models of human-computer interaction (HCI) and context concepts, then describes results of exploratory research to investigate the effect of knowledge about users, user tasks, organisational culture and user experiences on development of service-based applications. Results inform future research in this work-package through summaries that identify what types of HCI and context knowledge are more likely to effect different activities during the development and deployment of service-based applications.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany
University of Hamburg	Germany
VU Amsterdam	Netherlands

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL:

http://www.s-cube-network.eu/

Contents

List of acronyms

CoDAMoS: Context-Driven Adaptation of Mobile Services

CTT: ConcurrentTaskTree

EDDiE: Expansion and Disambiguation Discovery Engine: software tool

HCI: Human Computer Interaction

SBA: Service-Based Application

SECMOL: Service Centric Monitoring Language

SeCSE: Service Centric System Engineering

TCL: Task Constraint Language

UCaRE: Use Cases and Requirements: software tool

UML: Unified Modelling Language

UXM: User Experience Metric

WCAG: Web Content Accessibility Guidelines

Glossary

Accessibility: ability for individuals with diverse capacities, preferences and context of use, to use a product, a service or an environment – but not necessarily with the same degree of usability for all;

Assistive technology: technology enabling users with disabilities to perform functions that would otherwise be difficult or impossible for them;

Context: All elements outside the boundaries of a service-based application, for example the user, other systems, the physical environment, rules and regulations, belong to the context of the service-based application;

Context Factor: A context factor describes a set of elements in the context, which is relevant for discovering of services, for monitoring service-based applications and for adapting service-based applications;

Culture: patterns of thinking, feeling, and acting identified in a specific group (organization or nation);

Data collector: component that aggregates and produces the monitoring data against which constraints must be checked;

Facet: projection over one or more Service properties that provide a partial description of a service. They cover particular aspects of a service and can include Description, Signature and Quality of Service;

Facet specification: description of the service properties described by the facet in a given language;

Human Computer Interaction: study of the interaction between humans and computers;

Task modeling: description of the structured sets of activities that a user has to perform in order to attain goals;

Usability: the effectiveness, efficiency and satisfaction with that specified users can achieve specified goals in particular environment;

User interface: visible, physical representation of systems that allow users to interact with them and use the systems' functionalities. It is an abstracted description of the system's behavior, pared down to avoid overloading the user with irrelevant or unnecessarily abundant information;

User model: systems' representations of the properties of a user (such as his personal characteristics or preferences;

User modeling: process of generating models that systems have of users - system's representation of the user's characteristics that are necessary to adapt the system to this user's needs – that are kept within a computational environment.

1 Introduction

This deliverable reports research that intended to codify discipline knowledge from fields outside of software and service-oriented systems engineering, specifically considering HCI (Human Computer Interaction) and context, to understand how to engineer service-based applications for different and potentially changing and evolving usage environments. The results reported in the deliverable are designed to contribute to two long-term objectives of the JRA-1 work package:

1. Derive integrated principles, techniques and methodologies for engineering hybrid service based applications based on clearly identified design and discipline knowledge available in the functional service-based application layers and in research fields related to engineering service based applications, especially software engineering;
2. Contribute those principles, techniques and methods to the S-CUBE Integrating Framework (IA-3) and discipline knowledge to the S-CUBE convergence knowledge model (IA-1).

To produce the deliverable the authors set out to gather and coordinate design knowledge to develop new integrated models of how to engineer service-based applications. These models are descriptive, to enable researchers to compare and contrast processes, methods, models and analysis techniques. HCI and context knowledge has been used to build service-based applications to consider different user and task characteristics, drive service based application configuration, and support user-led personalization of services and service-based applications. The reported research seeks to overcome one identified weakness in service-based applications and their engineering – the omission of HCI and context knowledge from the engineering of such applications.

The results reported in this deliverable, codification of HCI knowledge relevant to service based application engineering, generates knowledge in forms that can be exploited in other activities. For example, codified knowledge about user tasks will be implemented as service discovery filters applied to existing service discovery engines in WP-JRA-2.3. Codified knowledge about user types will be implemented as service composition and coordination rules that take into account user characteristics in WP-JRA-2.2. And codified knowledge about user-driven customization will be applied in procedures for business process adaptation in WP-JRA-2.1. Research undertaken from month 11 onwards will be informed by results reported in this deliverable.

In compliance with the S-CUBE Description of Work, we undertook the following research to establish HCI knowledge relevant to service-based applications engineering:

1. Review related research literature and select formal task and user models with properties that represent codified knowledge about context factors associated with task and user characteristics pertinent to service based applications.

We have yet to undertake more focused research related to interaction design for service-based applications:

2. Review research into personalized user interfaces and multi-modal interaction to determine rules, patterns and guidelines for system and service-led configuration versus user-led customization of service based applications. In the future we will use a faceted classification scheme of context factors that can be applied to both consumer task and user models and extended specifications of services, thus providing a common underlying ontology of both services and their contexts.

The remainder of this deliverable is in 5 sections. Section 2 reports the results of review of 2 domains not reviewed in the original state-of-the-art deliverable ([1]) but deemed important to the production of this deliverable. Section 3 outlines our research method for codifying HCI and context knowledge to explain the results reported in the deliverable. Sections 4 and 5 report these results. Section 4 reports a series of meta-models that have been developed and validated to describe and model important HCI and context knowledge that influences the design and use of service-based applications. The models

provide an important input to S-CUBE convergence knowledge model. The models form the basis for section 5, which reports examples of codified knowledge that instantiates the meta-models. The codified knowledge is applied to important design-time and run-time activities associated with service-based applications: service specification, service discovery, service composition, and service monitoring. Section 5 results the codification of HCI knowledge. On the other hand we will report the analysis and codification of contextual knowledge from software engineering and service-oriented computing disciplines in the future JRA-1.1.4 deliverable. Section 6 reports how the meta-models and examples of instance-level codified knowledge will inform future research and model building in S-CUBE, in the form of simple lessons learned so far.

2 State-of-the-Art on Context and Organizational Culture

This section introduces 2 new areas that are reviewed for the first in S-CUBE: context in context-aware computing, and organizational culture. Both are reviewed here because both were not reviewed in the first 6 months of S-CUBE and were not reported in deliverable PO JRA.1.1.1. Both areas were identified as potentially relevant to S-CUBE because of ongoing research in these 2 areas. Hence their inclusion in this deliverable reflects a natural evolution of the exploratory research in S-CUBE.

Results from each literature review are reported in turn.

2.1 Context

This section reports previous work on context factors in context-aware computing and context factors in software engineering.

2.1.1 Context factors in context-aware computing

Our understanding of context related to mobile computing devices has evolved over the last decade. When mobile devices first became widely available, researchers treated the location of the device as the location of the device executing the software, such as the location of a mobile phone [2]. However, as research into mobile and ubiquitous computing matured, other important context factors were identified, such as other nearby users and devices, hosts and time [3] . More recently these context factors were organised systematically in ontologies. For example the Context-Driven Adaptation of Mobile Services (CoDAMoS) Ontology distinguishes between personal context (user), location context (the physical location) and device context (properties of the device, which executes the software) [4]

Another useful classification including various instances of the proposed classes was put forward by Chen and Kotz in [5]. The authors distinguish between the following types of contexts [5]:

- *Computing Context*: The computing context contains everything related to computational resources, e. g. available networks, network bandwidth, communication costs and nearby computational resources such as printers;
- *User Context*: The user context represents information about the user, which interacts with the software. This context includes information such as the user profile (age, preferences, etc.), the user's location (e. g. absolute position, indoors, outdoors, etc.) and orientation, nearby objects, the people nearby and the social situation;
- *Physical Context*: The physical context includes everything, which is measurable in the environment of the device with which the user interacts. The physical context includes temperatures, noise levels lighting situations, traffic conditions, etc;
- *Time Context*: The time context covers relevant information related to time such as absolute time, date, day of the week and season.

Chen and Kotz distinguish between two types of context awareness [5]: a software system, which exhibits features of *active context awareness*, adapts its behaviour to the context in which it operates. For example a navigation system adapts its behaviour (route planning) according to the context factor "traffic condition". In contrast to this active context awareness, *passive context awareness* means that the software system displays context information to the user or stores this context information for later retrieval. Consider, for instance, a GPS tracker, which continuously gathers location information and stores these data sets. Based on these data sets the user can reconstruct its way through a city. In this deliverable we only cover aspects of active context awareness.

As some user context such as the user profile, the user's tasks and its social environment (e. g. organisational culture) is at the heart of the HCI discipline, these aspects are described in Sub-section 2.2. What will be covered here is only one part of the user context namely the location context, which is currently the context factor widely used [5]. Consequently, we need to analyse the *computing context*, the *location context*, the *physical context* and the *time context (*similar in [6]).

Preuveneers et al. add more detail to the before-mentioned context factors in [7]. They distinguish the location context in relative locations (e. g. an address) and absolute locations (e. g. GPS coordinates). The authors further distinguish between the following physical context factors (called environmental condition in their work): temperature, pressure, humidity, lighting and noise. Lastly, the authors also specify the computing context (platform in their terminology) and distinguish between software (for example between operating system, virtual machine, middleware, etc.) and hardware. Hardware is further refined in resources (e. g. power, memory, CPU, storage, network, etc.) and devices (input and output devices). We use these context factors in our knowledge maps described in Section 4.

2.1.2 Context Factors in Software Engineering

In software engineering context factors are usually grouped according to different dimensions, worlds or facets (see [1] for the related work). Mylopoulos for instance distinguishes between the *subject world* describing the domain in which the software operates, the *system world* describing the system itself, the *usage world* describing the interaction of the user with the system and the *development world*, which describes how the software is developed [8]. As the development world will be discussed in CD-JRA-1.1.2, it will also be excluded here. What is described in this sub-section is the system world.

According to Pohl [9], the system world (called IT systems facet in his book) should include hard- and software components, which interact with the system in focus. *Services*, which are automated and made available as web services, can be relevant context factors. Service-oriented architectures assume an open-world view [10] in which services may not be under control of the service consumer, so service providers and consumers negotiate agreements about the quality of the service in service level agreements (SLAs) that are another context factor, and service discovery, monitoring and adaptation may be influenced by SLAs. Furthermore, service providers can update services provided to consumers without prior notice, leading to multiple versions of the same service being deployed and invoked.

Therefore, two context factors related to software engineering are multi-version services and service level agreements.

2.2 Organizational cultures

Although originally culture was a concept issued from anthropology, it has been described in many ways. Kroeber and Kluckhohn reported more than 300 definitions of culture [11]. In this section we define culture, identify the many types of layers and levels of culture, and discuss models and theories of culture. The final part of this section summarizes culture and the conclusions drawn from the literature review on the subject matter and suggests references to service-based architectures.

2.2.1 Definitions of Culture

In order to understand how culture can affect the use of software services, the definitions of culture must be examined. According to Hofstede, the majority of western languages interpret and associate culture with civilization and the refinement of the mind, for example, education, art and literature [12].

Choong [13] defines culture as the total patterns of human behaviour and the products embodied in thought, speech, action and artefacts. He further suggests that it is dependent upon the humans' capacity for learning and transmitting the knowledge to the next generation. Hofstede [12] reinforces this by adding that culture is learned, not inherited. He claims that it derives from one's social environment and not from one's genes. He further suggests that "*Culture should be distinguished from human nature on one side and from an individual's personality on the other, although exactly where*

the borders lie between human nature and culture, and between culture and personality, is a matter of discussion among social scientists".

Hofstede summarized culture as patterns of thinking, feeling, and acting identified in a specific group (organisation or nation). Therefore it can be derived that culture is not necessarily human nature even though it is determined by human behaviour. It is learned through the upbringing and surroundings of a certain individual's environment; it is not something that an individual is born with. Hence, it can be assumed that culture is a human mental programming of the mind. Figure 1 illustrates this.

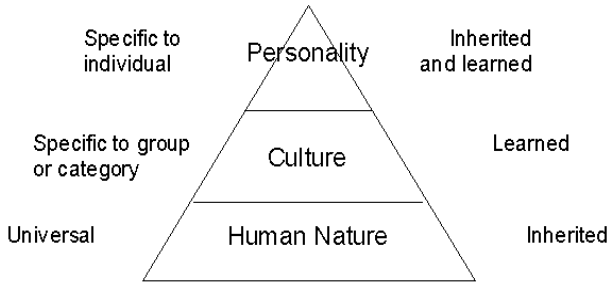

Figure 1: Three levels of uniqueness in human mental programming [5]

Del Galdo & Nielsen ([14]) added that history, language, and the level of technical development and nationality all affect culture. However, culture appears to be more tightly derived from the human sub-conscious rather than just the factors mentioned above.

2.2.2 Existing Theories and Models of Culture

In the last few decades there have been many theories and models that have been built to help establish and determine culture. Del Galdo & Nielsen ([14]) report that the most prominent researchers are Hall, Trompenaars, Hofstede, and Victor. The work of Hofstede and Trompenaars has studied at length, and Trompenaar's studies assimilated Hofstede's work. This section reviews Trompenaars and Hofstede's work as it is possibly the most influential in the field of culture.

2.2.3 Types and Levels of Culture

Social science researchers have divided culture into three main types:

1. National Culture
2. Organisational/Corporate Culture
3. Occupational/Sub/Professional Culture

Trompenaars ([15]) reports that the highest level is national culture (regional boundaries, for example Germany, Spain and France), then organizational culture - which is how attitudes are expressed in a specific organization - and finally professional and occupational sub-cultures, which is the particular culture within the departments of an organisation.

Hofstede's model/topology in Figure 2 illustrates that there are different levels of culture, shown on the left of the model. He reports that a person has already been culturally programmed by his/her family and by one or many schools, and has developed certain values to determine that culture. When entering a business environment, the cultural programming for the work places will have more to do with practices within the environment ([16]). In this deliverable we focus on organizational culture that is more relevant to the development and deployment of service-based applications.

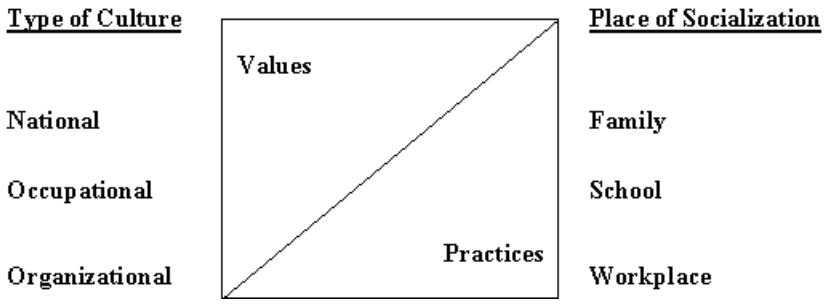

Figure 2: Hofstede's Levels of culture [16]

Hofstede states that organizational culture is acquired when we enter a work organization at a young age when our values are already rooted. This culture mainly consists of the organizations' practices [17]. Krumbholz ([18]) reviewed three different approaches to organisational culture, in which the work of Hofstede fits into the outside approach:

- **Metaphor approach**: culture as a metaphor where we are interested in how the organisation developed the culture;
- **Internal approach:** which emphasises on trying to find out how the culture can be improved;
- **Outside approach:** concerned with cross cultural comparisons and inter-cultural communication issues.

Moreover, organisational cultures also have sub-cultures within the many departments of the organisation. Trompenaars states it is "*The culture of particular functions within organisations: marketing, research and development, personnel. People within certain functions will tend to share certain professional and ethical orientations"* ([15]).

For example, in a University there could be different cultures in the Finance and Psychology Departments. Employees in different departments of an organisation will perform things differently. They may have different tasks, and perhaps achieve different goals, if this is so there will be different solutions, which in turn will affect the employees' values, beliefs and attitudes diversely.

2.2.4 Models of Culture

Figure 3 shows Hofstede's "onion" model of culture. Differences between cultures can manifest themselves differently as symbols, heroes, rituals and values [12]. Symbols are more external, observable manifestation of an organizational culture, whilst the underlying rituals and values of individuals and groups in the organization are no less important to determine the culture, but more difficult to observe.

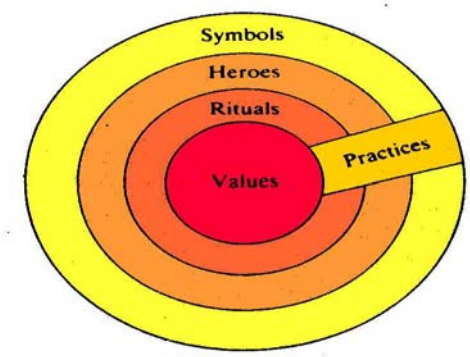

Figure 3: Hofstede's Model of Culture [12]

Trompenaars'' model of culture in Figure 4 is different from Hofstede's. He argues that on the outer layer are the explicit products, and that this layer consists of the explicit culture. According to Trompenaars, the explicit culture is *"the observable reality of the language, food, buildings, houses, monuments, agriculture, shrines, markets, fashions and art."*([15]). In the middle layer are the norms and values *"this layer reflects the norms and values of a group. Norms are what a group of people believe is right or wrong. It can be formally written as a set of rules and laws or informally stated. Values define good and bad. When the norms are correspondent to the values the group are relatively stable"*. Finally, a deeper, more implicit level of culture is represented at the inner layer. Trompenaars believes that this implicit culture is the basic assumption about existence. For him this is the core of culture as *"the problems of daily life are solved in such obvious ways that the solutions disappear from our consciousness."*[15].

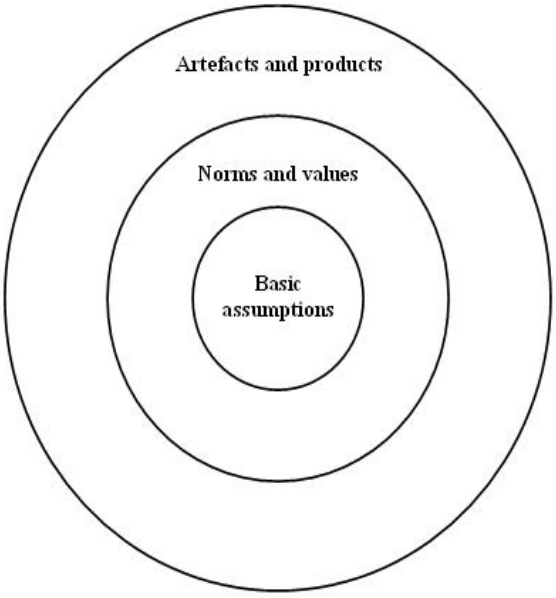

Figure 4: Trompenaars' Model of Culture [15]

Both Hofstede and Trompenaars describe dimensions between organizational cultures. These dimensions are reported in the next 2 sections.

2.2.5 Hofstede's Dimensions of Organizational Culture

Hofstede identifies 7 different dimensions of organizational culture: process- versus results-oriented; employee- versus job-oriented; parochial versus professional-dependent; open versus closed systems of communication; loose versus tight control; and normative versus pragmatic [12]. Each is described in turn.

- **Process- versus results-oriented:** process-oriented culture is an organisational culture that values how things are done; results-oriented culture values what gets done and the final outcome. Hofstede states: *"Process oriented - concern with means; people perceive themselves as avoiding risks and making only a limited effort in their jobs, each day is pretty much the same. Results oriented - concern with goals; people perceive themselves as comfortable in unfamiliar situations and put a maximal effort, while each day is felt to bring new challenges"*;
- **Employee versus job-oriented -** An organisational culture that is concerned about its employees and employee satisfaction versus an organisation concerned about the work, the job, and the employees capabilities. Hofstede states: *"Employee oriented - concern for people; people feel their personal problems are taken into account, the organisation takes a responsibility for employee welfare, and important decisions tend to be made by groups of*

committees. Job oriented - people experience strong pressure to complete the job, they perceive the organisation as only interested in the work employees do, no in their personal and family welfare, and important decisions tend to be made by individuals. N.B. Individuals can be both job and employee oriented at the same time, however organisational cultures tend to favour one or the other";

- **Parochial versus professional development -** Identity of employees taken from within the organisation versus identity taken from outside the organisation. Hofstede states: *"Parochial dependent - units whose employees derive their identity largely from the organisation; members of this culture feels the organisation norms cover their behaviour at home as well as on the job, they feel that in hiring employees, the company takes their social and family background into account as much as their job competence, and they do not look far into the future. Professional dependent - units in which people identify with their type of job; they consider their private lives their own business, they think the organisation hires on basis of job competence only, and they do think far ahead. Parochial culture is often associated with Japanese companies";*
- **Open versus closed systems of communication -** An organisation which is easy to join and employees can quickly get up to speed once they have joined versus an organisation which is difficult to join, where only specific types of people would fit in. Hofstede states: *"Open - In open systems units members consider both the organisation and its people open to newcomers and outsiders, almost anyone would fit into the organisation, and employees need only a few days to feel at home. Closed - In closed system units the organisation an its people are felt to be closed and secretive, even among insiders, only very special people fit into the organisation, and new employees need more than a year to feel at home"*;
- **Loose versus tight control -** These dimensions relate to the internal structuring of the organisation. Therefore, it is an organisation that improvises and is casual versus an organisation that is formal and punctual. Hofstede states: *"Loose control - units feel that no one thinks of cost, meeting times are only kept approximately, and jokes about the company and the job are frequent. Tight control - units describe their work environment as cost-conscious, meeting times are kept punctually, and jokes about the company and/or the job are rare";*
- **Normative versus pragmatic -** This dimension deals a lot with customer needs and orientation. Therefore, it is an ideologically driven versus a market driven organisation. Hofstede states: *"Normative - the major emphasis here is on correctly following organisational procedures, which again is more important than results, in matters of business ethics and honesty, the units standards are felt to be high. Pragmatic - major emphasis is in meeting the customers needs, results are more important than correct procedures, in matters of business ethics a pragmatic rather than dogmatic attitude prevails"*.

2.2.6 Trompenaars' Dimensions of Organizational Culture

Trompenaars reports 2 dimensions of organization culture - equality versus hierarchy and person versus task [15]:

- **Equality versus hierarchy –** This dimension is about the equality within the organisation versus the hierarchical structure of the organisation;
- **Person versus task –** This is an organisation concerned with the people doing the task versus an organisation concerned with the task.

Trompenaars dimensions are similar to 2 of Hofstede's dimensions. The equality versus hierarchy dimension is similar to Hofstede's loose versus tight control dimension, as both refer to the internal structuring and hierarchy of the organisation. Trompenaars' people versus task dimension can be mapped onto Hofstede's employee- versus job-oriented dimension, as both refer to an organisation's concern for the employees within the organisation versus the focus on task completion.

Conclusions
This extension of the original S-CUBE literature review has demonstrated that well-established models of culture exist, that these models are well documented and stable, and that these models can be used to understand the role of organizational culture in service-based applications. Results generated from some preliminary research in this direction are demonstrated later in this deliverable.

2.2.7 Internationalization and Localization of Web Services

This section explores current provisions for culture-related considerations in service engineering. As is apparent from section 2.3, research in culture and its relation with technology is long-established in HCI. Three terms are often used when referring to the application of such research to software engineering or the design of websites: Globalization, Internationalization and Localization.

According to [19], Globalization is *" the entire process of preparing products or services for worldwide production and consumption and includes issues at international, inter-cultural and local scales"*, while Internationalization is *"the process of preparing code that separates the localizable data and resources (that is, items that pertain to language and culture needed for input and output) from the primary functionality of the software".* The resources consulted on the topic of Globalization and web services however seemed to have a less well-defined distinction between Globalization and Internationalization, and so the remainder of this section will use the terms Internationalization and Localization in keeping with the following definitions:

- Internationalization is the process of engineering software that can serve the needs of users with differing language, cultural, or geographic requirements and expectations. This ensures that Web services have robust support for global use, including all of the world's languages and cultures [20, 21].
- Localization is the process of customizing the data and resources of code needed for a specific market (e.g. small scale communities, often with unified language and culture such as business or social organizations) [19].

Internationalization and Localization are interrelated in that the presence of Internationalization features permits an easier Localization of services. Considering that web services for the most do not have a user interface, and for many of them merely provide data that could then be formatted to the convenience of the user, the need for Internationalization features is not immediately clear. However, Internationalization as understood in HCI goes beyond data structures and formatting and can encompass various areas - the most obvious being the user interface (e.g. skins, colours, symbols, icons…), but also security (e.g. language-specific malware detection) or legal requirements to name but a few. As a result, the rethinking of the entire logic behind the composition of a SBA could potentially have to be performed depending on the intended user.

Cultural aspects can have a non-negligible impact on the adoption and use of services, as also expressed by Jonathan Schwartz, Chief Executive Officer and President of Sun Microsystems, Inc.: "It's becoming more true by the day that the globalization of network standards is allowing the *localization* of the internet itself. A web server in the US is the same as a web server in Brazil. But a web *service* based in the US is unlikely to succeed against its local Brazilian counterparts without comprehending local culture" [22].

The W3C acknowledged the importance of this issue by forming the W3C Internationalization Working Group that aims among other to permit the use of locale[1] and international preferences for the provision of internationalized and localized web service operations [23]. A stated goal is *"…to ensure that Web services have robust support for global use, including all of the world's languages and cultures".* As stated in their latest Working Draft, WS-I18N intends to be a building block that, in conjunction with other Web services protocols will among others provide a framework for globalization.

[1] Locale: collection of settings associated with a specific language, country, or market that embody the user's preferences [23]

An initial survey of resources found existing knowledge related to Cultural issues (to various extents), and applicable to the engineering of services:

- Internationalization patterns for web service deployment, applicable to the service or an aspect of the service [23]:
    - *Locale Neutral.* Most aspects of most services are not particularly locale-affected. These services can be considered "locale neutral". For example, a service that adds two integers together is locale neutral.
    - *Data Driven.* Aspects of the data determine how it is processed, rather than the configuration of either the requester or the provider.
    - *Service Determined.* The service will have a particular setting built into it. As in: this service always runs in the French for France locale. Or, quite commonly, the service will run in the host's default locale. It may even be a deployment decision that controls which locale or preferences are applied to the service's operation.
    - *Client Influenced.* The service's operation can use a locale preference provided by the end-user to affect its processing. This is called "influenced" because not every request may be honored by the service (the service may only implement behavior for certain locales or international preference combinations).
- Web Services Internationalization Usage Scenarios, intended as templates for Web Services Designers to implement Internationalization options [20].
- Requirements for the Internationalization of web service [23].

Despite the encouraging steps taken to address it, the challenge of providing Internationalization and Localization features for Web Services while retaining their platform independence is far from being solved; further development in this area may be supported by conducting research into Culture specifically for services.

2.2.8 Conclusions about Research into Organizational Culture

This additional literature review reveals that organizational culture and cultural dimensions are understood and stable in the relevant literatures. It provides a baseline for defining culture related to service-based applications, and for developing codified knowledge for designing and implementing these service-based applications.

2.3 Conclusions about State-of-the-Art on Context and Organizational Culture

This purpose of the additional reported literature was to extend the coverage of relevant context and HCI research domains above what was reported in the deliverable PO JRA1.1.1. Both topics were identified as relevant to future S-CUBE research directions. These directions are demonstrated with results from preliminary research in subsequent sections of this deliverable. Section 4 reports knowledge models that describe important associations between concepts about software services, context and organizational culture. Section 5 reports results from more grounded research that seeks to demonstrate important associations between organizational culture, business processes and service qualities during service selection and discovery activities.

3 Method

The research reported in this deliverable was developed using a simple method that was designed to demonstrate the feasibility and potential utility of codified HCI and context knowledge as quickly as possible to the S-CUBE consortium. Early concrete results were deemed important to communicate the role of HCI and context knowledge to researchers and research activities in JRA1 and JRA2. The method was in 6 main steps:

1. Review the results of the literature analysis ([1]) for the potential influence of different types of HCI and context knowledge on the design and deployment of service-based applications, extending the literature review into emerging domains as needed;
2. Select a subset of the different types of HCI and context knowledge with which to undertake a prototypical investigation of the codification of HCI and context knowledge for the design and deployment of service-based applications, to be reported in this deliverable;
3. Analyse the selected types of HCI and context knowledge and relate them with important concepts in the design and deployment of service-based applications through the construction and evaluation of UML meta-models;
4. Select one service-based application environment with which to develop and implement codified HCI and context knowledge;
5. Develop, implement and demonstrate the codified HCI and context knowledge for the implemented service-based application environment, then run a simple formative evaluation of the environments to explore the potential utility of the codified knowledge;
6. Reflect on the potential utility and cost of developing the codified HCI and context knowledge for designing and running service-based applications.

Each of the steps is described in more detail.

In steps 1 and 2 the review of the results of the literature analysis revealed the following types of HCI knowledge were most immediate and potentially useful to a service-based application:

1. User knowledge, often described in user profiles and dynamic user models, and representing relatively static attributes about a service user or consumer, such as age, and more dynamic attributes such the state of the user's current mental model;
2. Task knowledge, describing the user's task goals, actions, strategies and resources from the perspective of the user. Such fine-grain task knowledge is not often available to current service-based applications, and not reported in business process and work flow models;
3. Accessibility guidelines, containing standards and design advice to make products and services available to the wide range of users, including the disabled and the elderly. The accessibility to software-based systems is enshrined in EU and national law, and therefore is relevant to service-based applications. We reviewed established accessibility and standards and sought to codify how these might be implemented in service-based applications.

Furthermore we included a 4th type of HCI knowledge – knowledge about the organizational culture – and explored its potential consequences for the design and running of service-based applications. We chose to investigate organizational context knowledge to explore the broader potential benefits of HCI knowledge on service-based applications. The relationship between knowledge about an organisation's culture and its design and use of service-based applications was not immediately obvious. As such use of this knowledge acted as a boundary case, exploring the limits to the type of HCI knowledge that it might be useful to codify. We also reviewed knowledge about the software engineering context, and decided to include context-awareness from the ubiquitous systems community to explore the potential utility of codified context knowledge that is not HCI knowledge. Context-awareness includes familiar concepts including the location of the user, the user's proximity

to other users, places and locations, time, and the device upon which the service-based application is running.

In step 3 key concepts that are described, modelled and analyzed in user modelling, task modelling, accessibility standards and guidelines, organization culture analysis and context awareness were depicted using UML class diagrams, associated together, then related to existing meta-models of service-based applications such as the conceptual model of service-based systems developed by the FP6 SeCSE Integrated Project [24].

In step 4 we selected the SeCSE development and run-time environments for service-based systems upon which to design and implement codified HCI knowledge. The FP6 SeCSE Integrated Project [24] is one of the cornerstone research development projects in service-centric systems funded by the European Commission. It has produced substantial research, development and evaluation results, as well as tool suites available to be extended in S-CUBE. Several of the S-CUBE partners, including POLIMI and City University London, developed major components of the SeCSE environments, hence this familiarity provided the capability to rapidly extend these environments with codified HCI knowledge. The use of this environment demonstrates effective exploitation and development of research funded previously by the European Commission.

Research and development in SeCSE was divided into 4 major areas:

1. Service specification: methods and tools for use by service providers to describe and publish services so that these services can be discovered and invoked by service consumers;
2. Service discovery: methods and tools for use by service integrators and consumers to discover services throughout the development process and at run-time to rebind new services into a running service-based application;
3. Service-centric systems engineering: methods and tools for use by systems integrators to compose and orchestrate services into a designed service-based architecture that be deployed, and developing service-level agreements;
4. Service delivery: tools for use in run-time to support all run-time elements of service-based application, including service monitoring, provision and rebinding.

Therefore different HCI knowledge types, each identified for codification, were codified in different ways to support service specification, service discovery, service composition and service monitoring.

In step 5 we sought to implement the codified knowledge in the SeCSE environments. The following examples demonstrate the implemented knowledge. A new service facet was implemented to demonstrate compliance to accessibility standard guidelines during service specification. For service discovery we extended SeCSE service queries specified using XQuery and selection filters with types of knowledge about organizational culture and user task knowledge. For service composition we designed new service composition rules based on codified user knowledge. For service monitoring we designed monitoring patterns based on codified user knowledge to customize monitoring to individual user characteristics.

In step 6 the authors of the deliverable reflected on the potential of the codified and implemented HCI knowledge to inform further research and development in JRA1.

The next section reports these results in more detail.

4 Knowledge Modeling

The knowledge model described in this deliverable is represented as class diagrams of the unified modelling language [25]. Since UML was primarily made to model software systems and not for describing knowledge maps, we need to define what we understand by the constructs used in our diagrams (e. g. class). From the rich set of modelling constructs available for UML class diagrams we chose the following subset with the meaning described below (see [26] for the rationale of this approach):

- *Package*: The package is used to structure the knowledge map according to logical views. It is, therefore, only used to reduce the diagram size and to enhance the model's clarity. No additional meaning is ascribed to the package construct;
- *Dependency*: Dependencies are only used in the package diagram to describe high-level dependencies between the packages. These high level dependencies are further refined by the semantics of the packages' model elements;
- *Class*: A class represents a concept (term) of our knowledge model. Intrinsic properties, i.e. properties belonging only to this concept, are modelled as attributes. Mutual properties, i.e. properties representing a relation between concepts are represented as associations or aggregations (for the distinction between intrinsic and mutual properties see [27], p.222). We assign the following colour coding to our classes:
  - Classes with light grey background represent concepts with a high priority. This means that these terms will be used in JRA-1.1 to determine their impact on the discovery, adaptation and monitoring of services;
  - Classes with dark grey background represent terms with low priorities, e. g. they represent context factors, which will most likely not be considered in the near future;
  - The remaining classes have normal priority meaning that we will concentrate on them as the project proceeds;
- *Attribute*: An attribute represents an intrinsic property of a concept. It is used to refine the concept. All technical properties of attributes, e. g. data type, visibility, etc. are excluded from the knowledge model;
- *Associations*: Associations represent relationships between terms. A name of the association describes the relationship verbally. To enhance the readability of associations they might be directed (directed associations). This direction, however, does not assume any navigability;
- *Aggregations*: Aggregations are used to model a special relationship between terms where one aggregation end represents the whole and the other aggregation end represents the part. Aggregations need not be labelled with names;
- *Inheritance*: The inheritance relation is used to model the linguistic concept of subsumption (e.g. [28]), e. g. to express that one concept is more general then another. Inheritance relations between classes should be modelled top-down whenever possible to enhance the readability of the diagram [29];
- *Association Classes*: Association classes represent the concept of property precedence [30, 31]. Hence, association classes are used to express that a concept (association class) can only exist if the relation (association) between two concepts (classes) exists;
- *Stereotypes*: Stereotypes are currently only used for classes. To represent visual cues between different model views (diagrams), some classes are present in different diagrams [32]. The stereotype is used for these classes to indicate the package in which these classes are defined.

The knowledge model of the context factors are organised in four different views (cf. Figure 5):

1. *Consumer Package*: The central element in the model is the consumer package. It represents the knowledge model of service consumers. Consumers use services of different service providers. This service selection may be influenced by the culture of the organisation in which the consumer is embedded (dependencies consumer-culture and culture-provider). The consumer with its tasks, attitudes and its organisational culture also influences the engineering of SBAs (dependencies consumer-engineering and consumer-culture-engineering);
2. *Provider Package*: The provider package represents the knowledge model of service providers. Service providers are influenced directly and indirectly (via the organisational culture) by service consumers (see above). In addition, service providers influence the engineering of SBAs, e. g. by offering new services and/or new service level agreements;
3. *Engineering Package*: The engineering package represents the knowledge model of the systems integration of the provider, culture and consumer views during the engineering process. It, therefore, describes how the consumer, provider and the organisational culture influence the way in which SBAs are engineered, monitored and adapted.
4. *Culture*: The culture package represents the organisational culture of the consumer's organisation. It is used to identify important cultural aspects. Since the organisational culture is not specific for one service consumer, it is modelled as separate package in our knowledge model.

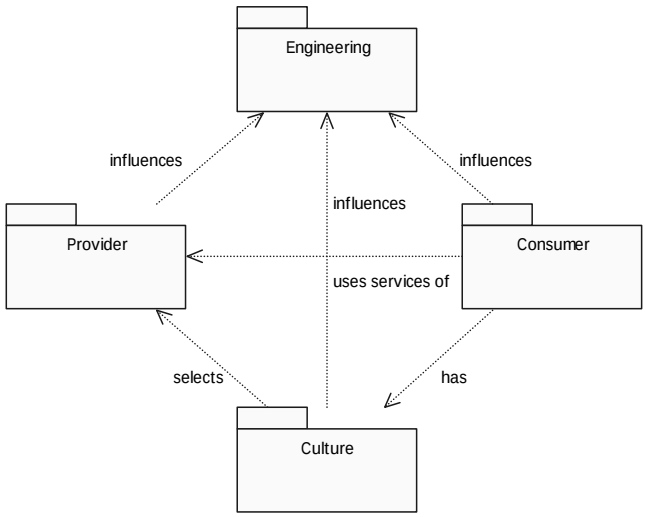

Figure 5: Views of the Knowledge Model

As in the UML the integration between the different packages is achieved by using the same classes in more than one package. In this case, the stereotype of the respective class is used to describe the origin of that class. To enhance the readability of the deliverable document, these references to other packages are described using the following notation [➔ Model element (cf. Figure to the model)].

The content of the four packages is described in the following subsections below.

4.1 Consumer View

The service consumer view is used to describe context factors related to all aspects of the service user, often also referred to the service consumer. In this knowledge model we distinguish between the following four context factors (cf. Figure 6 and their detailed description in Figure 7):

1. *ComputingContext:* The ComputingContext is distinguished into soft- and hardware. The software functionality is provided by services [➔ This element is also integrated into the Consumer View Model below (cf. Figure 7)].
2. *PhysicalContext:* As PhysicalContext we regard measurable factors such as temperature, pressure, humidity, lighting and noise [➔ This element is also integrated into the Consumer View Model below (cf. Figure 7).]
3. *LocationContext:* For the LocationContext we distinguish between a location and an orientation (meaning a given direction between two locations). Locations may be absolute (e. g. GPS coordinates) or relative (e. g. an address). It is important to note that the user and the device have a LocationContext and a PhysicalContext. This means that different context factors may be relevant for the user and for the device [➔ This element is also integrated into the Consumer View Model below (cf. Figure 7).]
4. *TimeContext:* The TimeContext specifies when the user uses the SBA or when a service is been executed. Time in this sense can be absolute or relative (e. g. the day of the week) [➔ This element is also integrated into the Consumer View Model below (cf. Figure 7)].

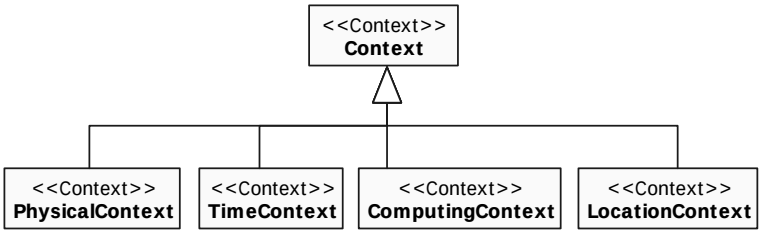

Figure 6: Knowledge Model of the Context Factors

The above-mentioned context factors are important because they may influence the interaction of the user with the system and/or the provision of services to the user. Consider for instance a mobile navigation device. Its services may decide to present the user with relevant background information about a certain sight if the device is situated a certain location (LocationContext).

In the knowledge model we also distinguish between three different HCI context factors:

1. *User*: The user is a service consumer. This service consumer may be a person within an organisation. [➔ The concept of Person is also integrated in the Culture View (cf. Figure 9)] The user concept represents the person or institution, which uses the service in a specific *physical*, *temporal* and *spatial* context as well as in a specific *social setting* and on a specific *device* (associations User-PhysicalContext, User-Time, User-SocialSetting, User-LocationContext, User-Device). The aim of the user is to carry out some *task* (association User-Task). For each of those *tasks* the user assigns a goal (association class UserGoal);
2. *Task*: The task represents the activity the *user* has to perform in order to fulfil its goals. In SBAs these tasks should be supported by services. Each task can be decomposed into subtasks (reflexive association of class Task). In HCI and in particular in task modelling we distinguish between abstract tasks, which are further decomposed, user tasks – tasks to be carried out by the user –, interaction tasks – tasks to be carried out by the interaction of a user with a software system – and application tasks representing those tasks, which are fully supported by software system. The sequence of different task is described by operators (class SequencingOperator and association SequencingOperator-Task) [➔ The SequencingOperator is also used to describe business processes and is, therefore, integrated in the Engineering View (cf. Figure 10)];

3. *Accessibility*: The accessibility class models the relation between the device used by the user to access the service and the software interface design (classes Device, SoftwareInterfaceDesign and associations Accessibility-Device, Device-SoftwareInterfaceDesign, SoftwareInterfaceDesign-Accessibility).

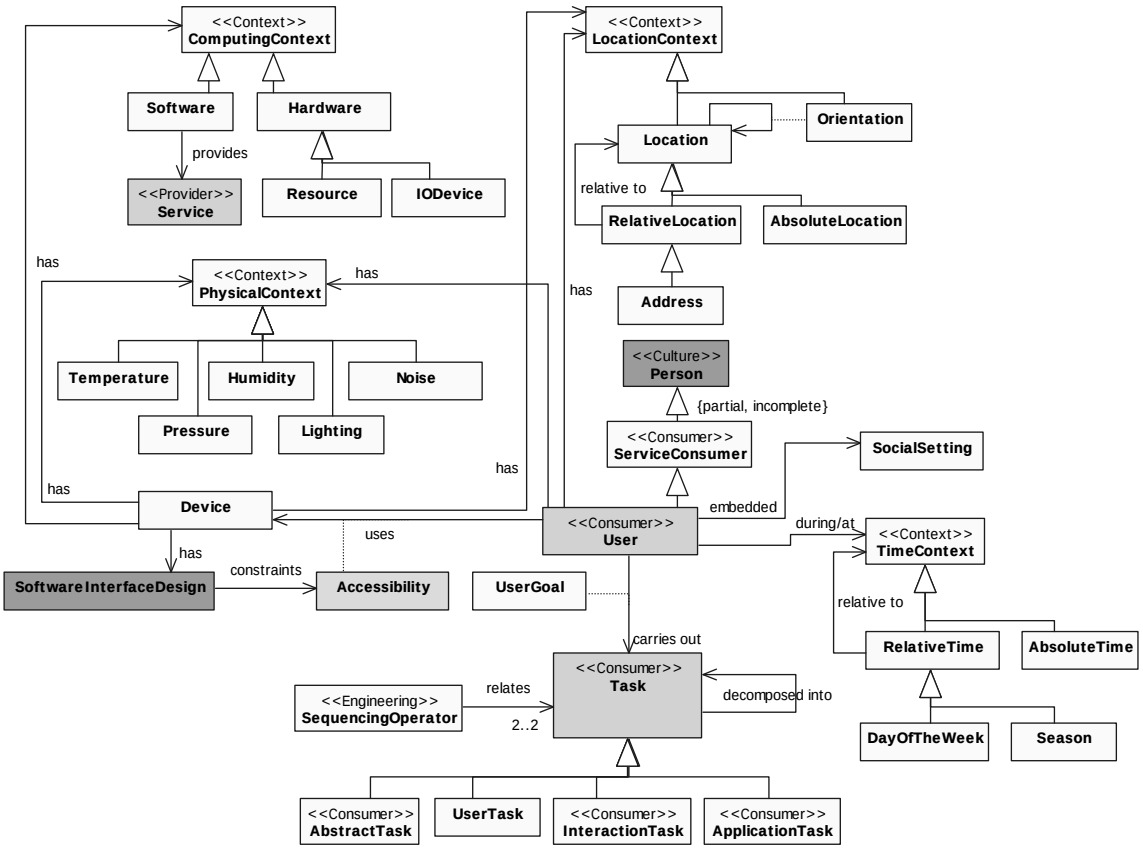

Figure 7: Knowledge Model of the Consumer View

4.2 Provider View

The knowledge model of the provider view is presented in Figure 8. We were able to identify four different context factors:

1. *ServiceProvider*: The service provider is the person or organisation, who offers the service;
2. *Service*: The service is the central element in the knowledge model of the service provider. It represents the *functionality* offered by the *service provider* for a certain set of *quality attributes*. *Services* are important context factors in continuous requirements engineering processes since new services can be used to optimise a given SBA [33]. Each service has is embedded in some context (association to class Context) [➔ The class context is integrated into the Consumer View (cf.. Figure 6)];
3. *Quality Attribute, Functionality and FunctionDescription:* A quality attribute describes characteristics, e. g. response time of a certain functionality of the service ([34]), association QualityAttribute-FunctionDescription-Functionality);
4. *SLA:* The service level agreement is a contract between a service provider and a service consumer, which contains a description of the functionality offered including the qualitative description of the functionality (aggregation SLA-FunctionDescription, association

ServiceProvider-ServiceConsumer, association class SLA) [➔ The ServiceConsumer is an element of the Consumer View and is integrated there (cf. Figure 7)].

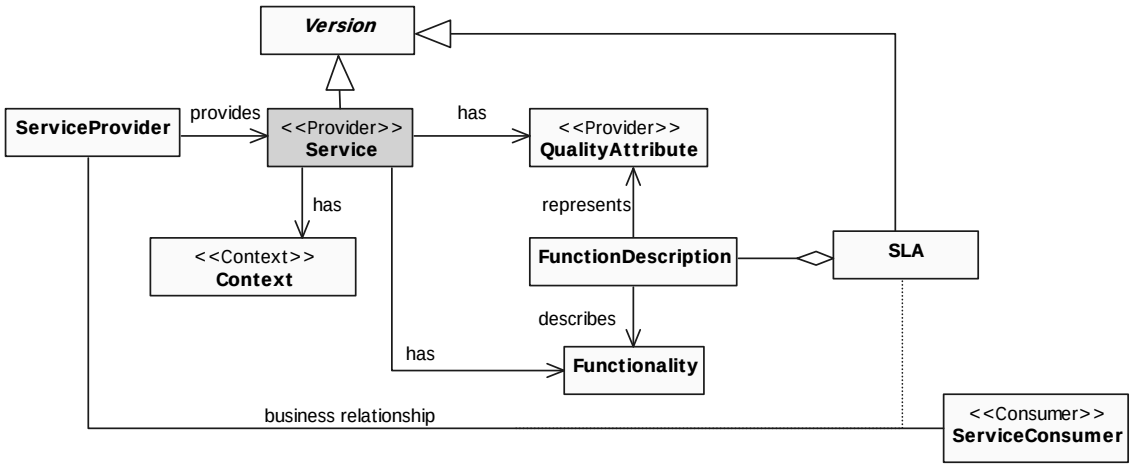

Figure 8: Knowledge Model of the Provider View

The abstract class Version represents the fact that services and service level agreements are updated over time, for example to enhance the functionality of the service and/or to enhance its quality parameters. Tracking these versions is important for service discovery and the adaptation of service compositions.

4.3 Culture View

The central element of the culture view is the organisational culture (cf. Figure 8), which can be instantiated using different culture dimensions. Organizational culture can be categorised in 6 dimensions, namely process vs. result oriented organisations, parochial vs. professional dependencies, lose vs. tight control, normative vs. pragmatic, employee oriented vs. job oriented and open vs. closed systems of communications. We represent these dimensions as attributes of the class OrganisationalCulture).

These dimensions of organisational culture are an aggregation of an organization's norms, values and beliefs (see respective classes in Figure 8), and are impacted on by the organisational structure of the enterprise, as expressed by the association OrganisationalStructure-OrganisationalCulture. The cultural aspects are embodied in people who work in the enterprise represented in Figure 8 by the classes Person and Workplace.

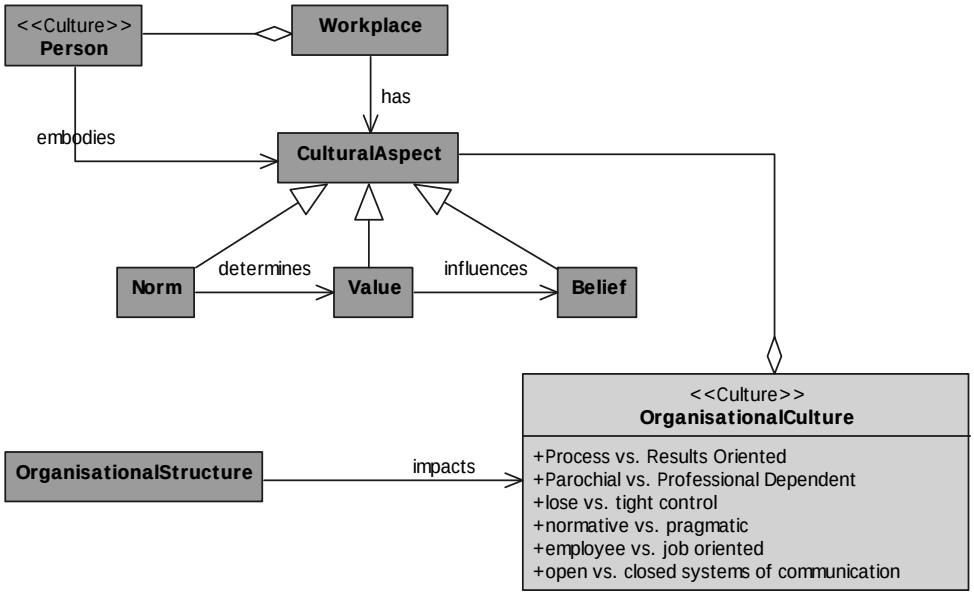

Figure 9: Knowledge Model of the Organisational Culture View

4.4 Engineering View

The engineering view depicted in Figure 10 describes how the context factors of the service consumer, service provider and culture view inform the design and use of service-based applications. To analyse this influence, we assume that the SBA is a process-oriented application, e. g. it is based on a workflow or business process. To execute the business process, it invokes different services. Together with the elements contained in all other views we can distinguish four central elements in the model:

1. *BusinessProcess*: The business process is a sequence of activities, which is described by its control flow (class SequencingOperator and association SequencingOperator-BuinessProcess). The process may be affected by the available services (see below; associations BusinessProcess-Service);
2. *QualityOfProcess:* The quality of process element describes certain quality characteristics of the business process (Association BusinessProcess-QualityOfProcess). These quality characteristics are in turn described by quality attributes (class QualityAttribute and association QualityAttribute-QualityOfProcess) [➔ Quality attributes are further described in the Provider View (cf. Figure 8)]; Quality characteristics of processes may guides the selection of services that fit best with the organizational culture (association QualityOfProcess-Service) [➔ The element OrganisationalCulture is integrated in the Culture View (cf. Figure 9)];
3. A link between the organisational culture and the quality of process can be established (association OrganisationalCulture-QualityOfProcess), which then guides the selection of services to be used in the business process and, therefore, also indirectly affects the business process;
4. *Service*: Services are used to realise business processes (association "invoke" BusinessProcess-Service) [➔ The Service class is integrated in the Provider View (cf. Figure 8)]. As new services become available, the business process may be changed to exploit the full functionality of these new services (association "affects" BusinessProcess-Service". From the HCI perspective a service may either be realised by application tasks or by interaction tasks (association InteractionTask-Service, ApplicationTask-Service and respective classes) [➔ Both classes IntegrationTask and ApplicationTask are further refined in the Consumer View (cf. Figure 7)].

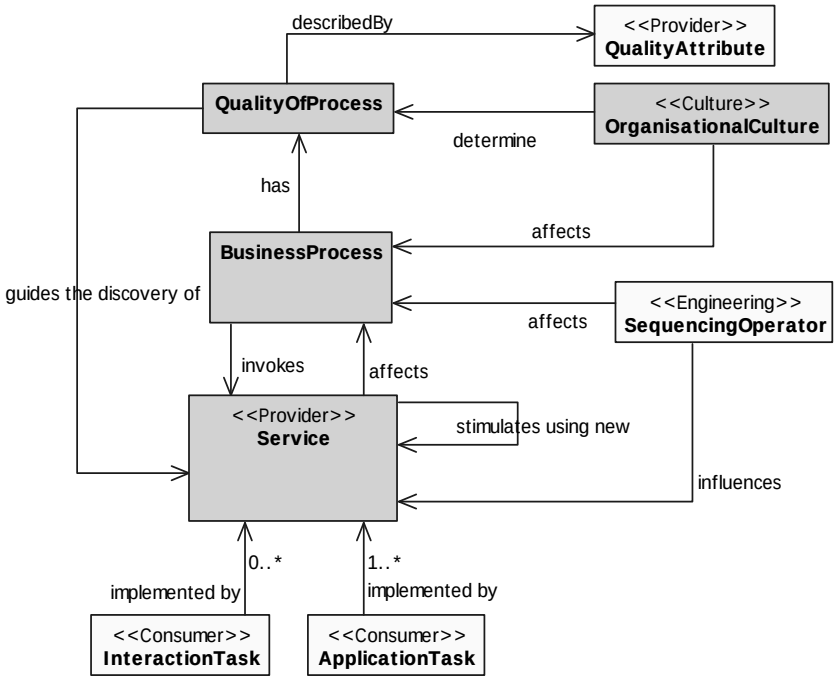

Figure 10: Knowledge Model of the Engineering View

4.5 Knowledge Modeling Conclusions

This section has scoped different context factors that impact on the design and delivery of service-based applications. These context factors have been grouped into 4 themes that are the engineering, provider, consumer and culture themes. For each theme we have defined and modelled the associations between important elements that represent these context factors. These models draw on previous research in related domains and reported conceptual models in service-centric systems research.

The next section elaborates on some of these context factors. It summarizes work undertake between months 7 and 10 of S-CUBE to explore how to codify HCI knowledge about service consumer tasks, service consumers in the form of user models, the organizational culture of these service consumers, and accessibility standards and guidelines. The reported research sought to explore the possible effect of the codified knowledge on the development and use of service-based applications. Demonstrating this possible effect is critical to deriving integrated principles, techniques and methods for engineering hybrid service-based applications based on codified HCI knowledge. At this relatively early stage in S-CUBE, the research is preliminary and exploratory, and not all lines of research reported in this section will be pursued in the remaining 3 years of S-CUBE.

5 Codified HCI Knowledge

Codified HCI knowledge can be exploited at several stages of the service-based application lifecycle to enhance service-based application development. In the deliverable, as an initial discussion, we explore codification of HCI knowledge about service consumer tasks, service consumers in the form of user models, the organizational culture of these service consumers, and accessibility standards and guidelines. Figure 10 depicts an overview of the lifecycle model developed in S-CUBE extended with the potential uses of this and other codified HCI knowledge.

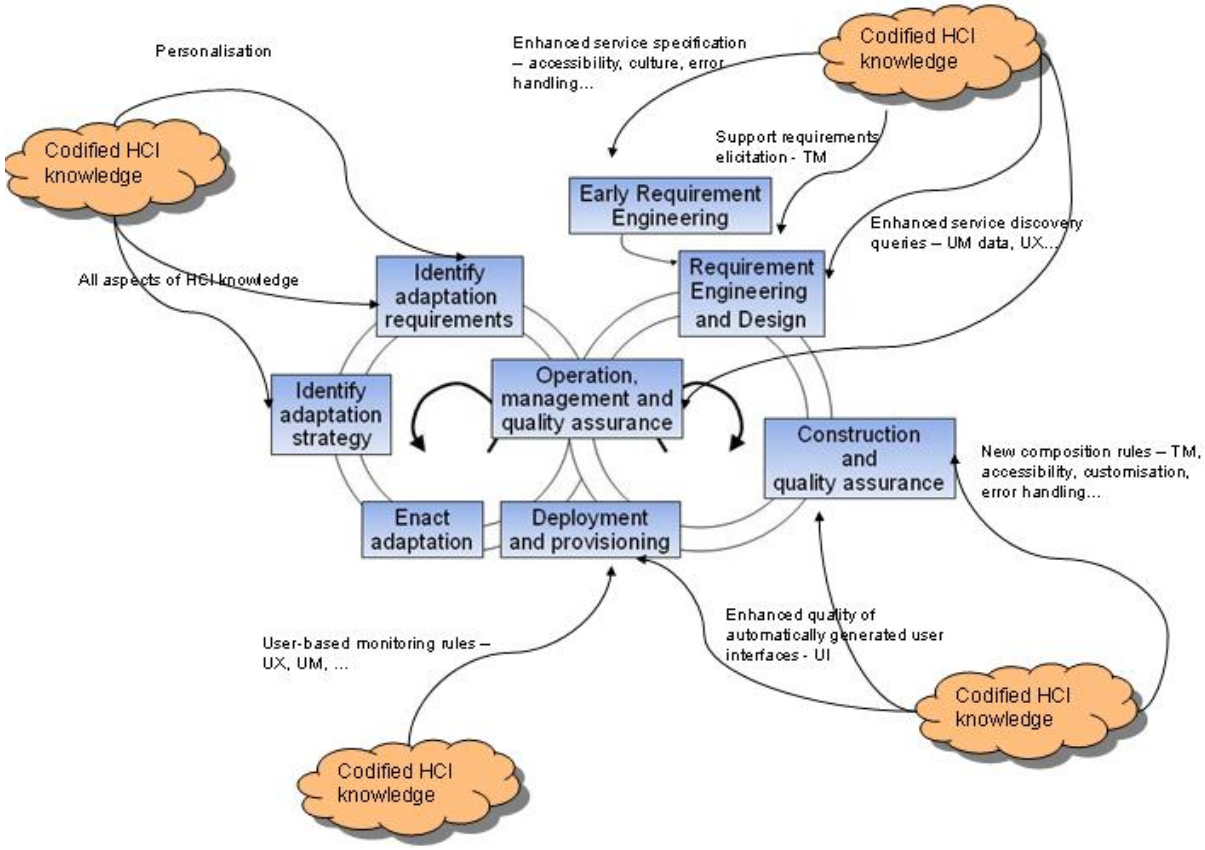

Figure 11: the S-CUBE method enhanced with codified HCI knowledge

We envisage possible contributions of codified HCI knowledge in the following phases:

1. **Early requirement engineering/requirement engineering and design**: the codification of knowledge from user task models is expected to support design and requirement elicitations activities, whilst codification of HCI knowledge about accessibility standards and organizational culture can provide meta-data to enhance service specification and heuristics that inform service selection;
2. **Construction and quality assurance**: codification of HCI knowledge about, for example task models, have the potential to generate service composition heuristics, and knowledge that codifies user interfaces can inform automatic generation of interfaces that are more usable and useful to human service consumers;
3. **Deployment and provisioning**: codified HCI knowledge can be applied to develop user-based monitoring rules using knowledge from user models and documented user experience models. Again, in the future, automated interface generation effected can also be supported;

4. **Operation and management**: as well as provided additional knowledge to inform service monitoring rules and strategies, codified HCI knowledge can be applied to refine run-time service discovery queries using knowledge about users and user tasks;
5. **Identify adaptation requirements/identify adaptation strategy**: many different types of codified HCI knowledge identified in deliverable PO JRA 1.1.1 can inform activities these two phases by providing additional, user and task-based knowledge. In particular knowledge about user-led personalisation may support the identification of adaptation requirements.

In the remainder of this section we report results from 4 separate areas of research that explore how to codify different types of HCI knowledge, and what effects that codified knowledge can have on the design and use of service-based applications.

5.1 Service specification

This section reports the results of codification of HCI knowledge about the accessibility of services to the task of service specification – developing the description of services properties to permit an informed evaluation of their suitability for particular purposes by consumers.

In the field of HCI, accessibility refers to the ability for individuals with diverse capacities, preferences and context of use, to use a product, a service or an environment, although not necessarily with the same degree of usability for all ([35, 36]). Principles, standards and guidelines exist to support the design of accessible software and web content (e.g. those issued by the World Wide Web Consortium). According to the UK Disability Discrimination Act [37], compliance with relevant accessibility guidelines and standards is a legal requirement for services (in the general sense of the word), a provision that is extensible to services provided over the web and hence to web services.

From a consumer point of view, the accessibility of a service (where applicable) is an important property to know about since it can be a decisive factor for service selection – in the case of disabled or older users for instance. As a consequence service providers seeking to widen the range of their services' users would benefit from supplying clear information about the accessibility of their services. One way to achieve this could be to describe the accessibility of services using a faceted approach. An overview of faceted service specification from SeCSE described in [38] is presented below.

Faceted service specification describes a service using both a standard UDDI (Universal Description, Discovery and Integration) specification and an optional set of *facets*. Facets are projections over service properties. They describe service properties in *Facet Specifications* using *Languages* that may be natural (e.g. English) or XML based (hence permitting both human and computerized interpretation of the specification) as depicted in the conceptual model in Figure 11. Existing core types of facets include Commerce, Management, Testing and QoS.

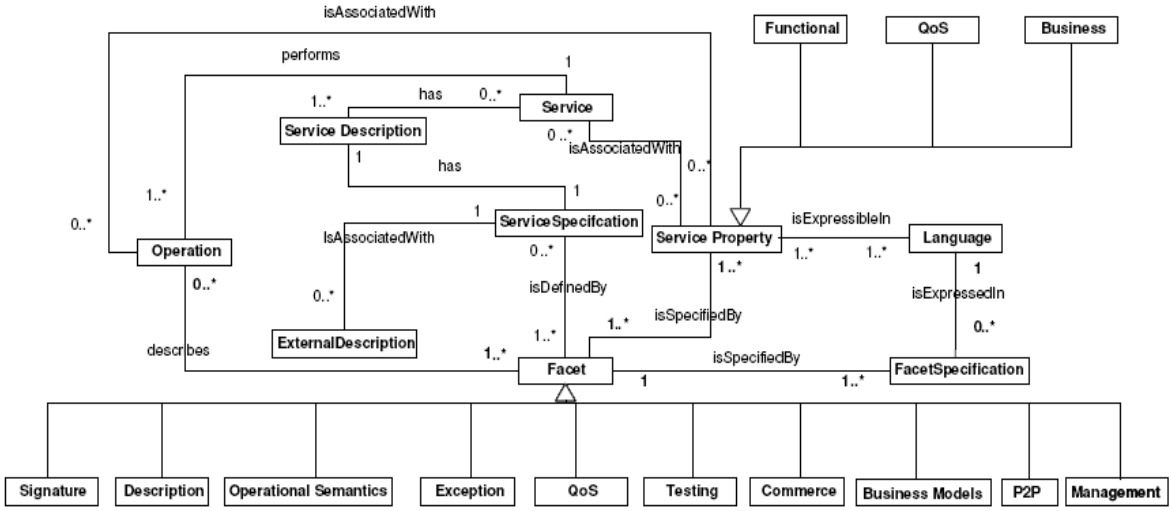

Figure 12: Faceted specification conceptual model [38]

The structure of a service specification file in the faceted approach is as follows:

The *Service specification* element includes:

- The service name;
- The service ID (a unique identifier);
- The service specification's last revision date;
- The service's available facets list (optional);
- Links to external specifications of the service.

Each *facet* element must include:

- The facet type;
- The facet owner or author;
- The list of the available facet specifications.

In turn, each *facet specification* element includes:

- The facet specification language;
- A link to the file containing the Language Specific Specification;
- A reference to any Ontology needed to interpret the Facet Specification (optional);
- A reference to any Service Information Model (SIM, where the data types used by the developers can be defined) needed to interpret the Facet Specification (optional).

Finally, the *Language Specific Specification* element includes:

- The facet type;
- The facet specification language;
- The facet specification owner or author;
- The facet specification's last revision date;
- The facet specification text (facet specification data);
- A reference to any Ontology needed to interpret the Facet Specification (optional);
- A reference to any SIM needed to interpret the Facet Specification (optional).

It can be noted that both service providers and consumers can exploit the data redundancy in the facet specification file. The service providers can help manage the Facet Specifications and the Service Consumer can verify the correctness of the relationship between the Facet and the Language Specific Specification relationship.

Figure 12 depicts the service specification file structure described above. The optional elements have an asterisk (*) appended to them.

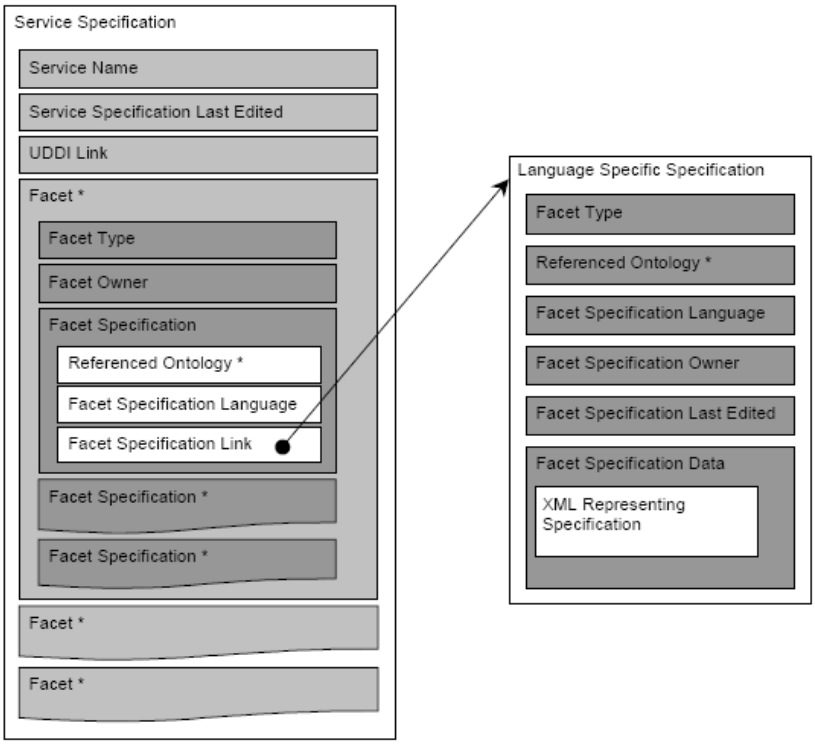

Figure 13: Faceted Service Specification file structure [39]

In keeping with the reported approach S-CUBE prototyped a new specification facet called the *accessibility facet*. This facet, depicted in Figure 13, permits the specification of:

- The accessibility guidelines and standards complied with by a service;
- The assistive technology compatible for use with the service;
- The provisions made regarding the general accessibility of a service or specific impairments/impairment types.

Assistive technology here refers to devices that allow impaired users to perform tasks they could not otherwise do, or support them in performing tasks with a greater ease. An example of such technology is screen readers that try to identify and describe the content displayed on a computer screen. They can read out content or, for some, output it in Braille; they are mainly used by visually impaired users, or users with learning or reading difficulties.

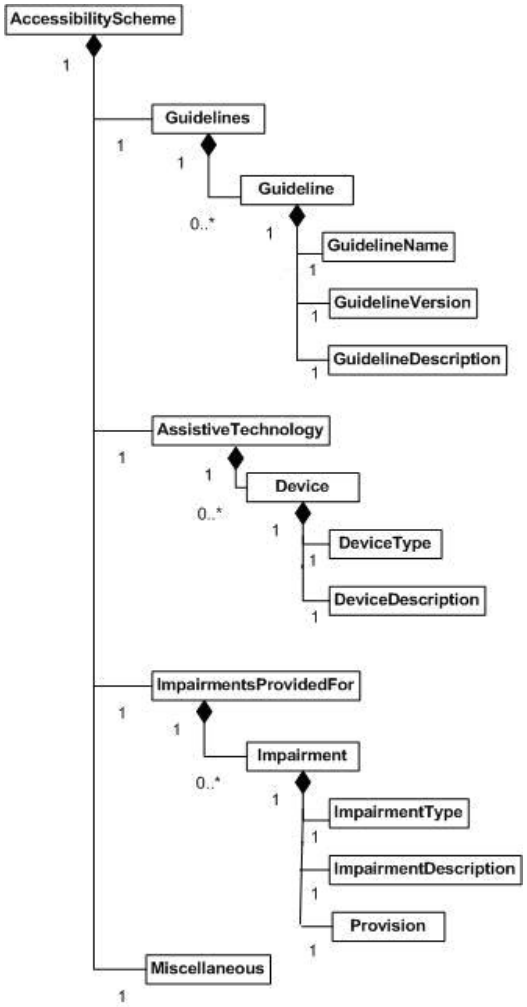

Figure 14: Accessibility facet

The corresponding facet specification data is as follows:

```
<FacetSpecificationData>
   <AccessibilitySpec>
      <Guidelines> - a list of the guidelines adhered to
         <Guideline> - multiple guidelines specifications can be created
            <GuidelineName/> - Name of the guideline
       <GuidelineVersion/> - version of the guideline followed
       <GuidelineDescription> - description of the guideline
          </Guideline>
       </Guidelines>
       <AssistiveTechnology> - list of compatible assistive technology
          <Device>
       <DeviceType/> - alternative input device, reading tool etc.
       <DeviceDescription/> - description of the device capabilites
          </Device>
        </AssistiveTechnology>
       <ImpairmentHandled> - list of the impairments the service is accessible for
          <Impairment> - multiple impairment specifications can be created
       <ImpairmentType/> - motor, cognitive, auditory, visual, etc.
       <ImpairmentDescription/> - description of the impairment
       <Provision/> -  provision made for the impairment
          </Impairment>
       </ImpairmentHandled>
       <Miscellaneous/> Any additional accessibility related information
   </AccessibilitySpec>
</FacetSpecificationData>
```

Considering the case of a visually impaired user seeking to use a service for example, we could envisage data from the accessibility facet specified above being used for service selection. Important

selection criteria for this particular user could be compatibility with a screen reader, compliance with specific guidelines related to web content access for the blind/visually impaired, or handling of his impairment in any other ways (e.g. the service output, if any, is auditory rather than text-based). Having this information readily available in the service description would support an informed selection between available services that have similar core functionalities, but differing accessibility levels for sight-impaired users.

To demonstrate and provide a first verification of the new accessibility facet, the facet has been instantiated to include specification of two W3C accessibility guidelines, two devices including a screen reader to support visually impaired users, and one type of impairment handled – a sight impairment.

```
<FacetSpecificationData>
   <AccessibilitySpec>
      <Guidelines> WCAG, XAG
         <Guideline>
             <GuidelineName/> WCAG
       <GuidelineVersion/> 1.0
       <GuidelineDescription> W3C Web Content Accessibility Guidelines; promote web content
       accessibility independently of the user agent used
          </Guideline>

       <GuidelineName/> XAG
       <GuidelineVersion/> 1.0
       <GuidelineDescription> W3C XML Accessibility Guidelines. Support the inclusion, in XML
       applications, of features that promote accessibility for users with disabilities.
          </Guideline>
       </Guidelines>
       <AssistiveTechnology> Text-to-speech screen readers, refreshable Braille display.
          <Device>
       <DeviceType/> Refreshable Braille display
       <DeviceDescription/> Convert textual standard output to Braille.
          </Device>

          <Device>
       <DeviceType/> software screen reader (text-to-speech)
       <DeviceDescription/> Convert textual standard output to audio. Tested supported screen
       readers are JAWS 7.0 and above, NVDA, ZoomText 8.0 and above.
          </Device>
        </AssistiveTechnology>

       <ImpairmentHandled> Sight impairment
          <Impairment>
       <ImpairmentType/> sight impairment
       <ImpairmentDescription/> Total or partial sightloss
       <Provision/> compatible with screen reader and Braille display.
          </Impairment>
       </ImpairmentHandled>
    <Miscellaneous/> Partially complies with WAI-ARIA 1.0: complies with the normative
    requirements related to WAI-ARIA roles (including user input widgets and user interface
    elements).
   </AccessibilitySpec>
</FacetSpecificationData>
```

The codification of accessibility knowledge into a facet presented in this section is a preliminary example of codified HCI knowledge applied to service discovery. Challenges still remain to be addressed, such as a systematic identification of services' properties that are influential for their accessibility, yet independent from their implementation (e.g. input/output data format independence). This example however gives an insight into one potential use of codified HCI knowledge and its possible applications for service selection. The next sections explore other possible uses.

5.2 Service discovery

Service discovery is a critical challenge in service-based applications. During the development of service-based applications the functionality and architecture of the application are informed by the services that are available, and these services need to be discovered. During the use of service-based applications new services need to be discovered if these service become available or currently invoked services need to be replaced by other services with improved qualities such as performance and reliability. Processes and techniques for service discovery have been researched extensively in previous projects [SeCSE references]. However, none of these processes and techniques explicitly use

HCI knowledge. In this section we report the results of preliminary research that explores the use of codified HCI knowledge during service discovery.

In this section we report 3 research results that explored different types of codified HCI knowledge:

1. Knowledge about user tasks, to refine the discovery and selection of services appropriate to the user task;
2. Knowledge about the user's profile and characteristics, to refine discovery and selection of services appropriate to the individual service consumer;
3. Knowledge about the organizational culture, to selection between services with qualities better aligned to the characteristics of the organization.

Results from each of these activities are reported in turn.

5.2.1 CTT Task Model Patterns for Query formulation

This section reports the application of task models as part of a new requirements-based service discovery approach that we are investigating in S-CUBE. It describes how task model patterns enhance the service discovery algorithms based on a Task Knowledge Base (Task KB) developed from S-CUBE research.

SeCSE's current service discovery environment implements an algorithm for discovering services based on requirements specifications using query expansion and word sense disambiguation techniques [40]. However, query expansion alone cannot resolve the semantic mismatch problem that arises because the problem request and solution service are inevitably expressed using different ontologies. To overcome this ontological mismatch, we are extending the algorithm with pattern libraries that encapsulate knowledge about classes of proven service solution to classes of user tasks. As such, task-oriented service discovery supports the user in finding appropriate services by querying a rich Task KB containing knowledge and models about typical tasks. These tasks can be both similar and complex. The task models and the semantics used to represent these models represent codified HCI knowledge not normally available to service discovery for service-based applications.

Our research uses Alexander's original definition of a pattern as a proven solution to a recurring problem in a context of use ([41]). In S-Cube we employ this definition to describe: (i) classes of tasks that re-occur during the design of service-centric applications, and: (ii) classes of candidate service solutions proven to solve these tasks. To decouple pattern development from the publication of concrete software services by service providers in distributed and heterogeneous service registries, task patterns do not reference concrete services in these registries. Instead, each pattern specifies classes of service that transform the service query and are matches to discover instances of new software services in service registries.

We envisage that task model patterns will:

- Contain descriptions for abstract as well as concrete tasks and their interrelations as semantic descriptions that have the potential to be compliant with requirements that are instantiated as service queries during early service discovery;
- Define task-specific categories that are compliant with classes of software service.

5.2.1.1 Populating the Task Knowledge Base

One goal of this research is to create task ontologies for modeling real world user activities. To avoid the ontology-modeling bottleneck that often inhibits ontology-based solutions we are seeking to extract task knowledge that can be reused. Our approach is to identify task knowledge that is domain-specific, then extract the domain-independent task knowledge that can be reused, similar to the KADS approach to knowledge modeling. For instance, domain-independent task knowledge that describes how to "go to *somewhere*", describes a general process model to perform the activity of moving from a starting point to a destination, is common knowledge among specific task knowledge regarding going to specific places from specific places. Such domain-independent task knowledge can be used to

describe more specific task knowledge in a new domain that decreases the cost for expanding the coverage of task knowledge.

5.2.1.2 Modular Task Models

In order to contribute to the creation of the Task KB, we propose creating task models in a modular fashion. More precisely, higher-level models are created through composition of lower-level task models. This becomes possible if we define a task model as a task tree, whose leaves are either *atomic tasks* or *references to other task models.* We suggest breaking down the overall task model into "sub-task models" which are of manageable size and are reusable in different contexts of use. Examples of generically applicable (sub) task models are "Book", "Find", "Request", etc. In that way we are able to define tasks that frequently occur in the design of service-centric applications.

In order to visualize the high-level structure of task models, we adopt the graphical notation for "Task-Model Diagrams" [42]. A task-model diagram conveys the structural properties of task models by highlighting relationships defined among them. Within a simple task-model diagram (Figure 14), task models are depicted by ellipses and their relationships are visualized using arrows and lines. Two relationships exist: *Include* and *Specialization*. The former is labeled "include" and it denotes the hierarchical composition of high-level task models from lower-level task models. The high-level task model "Travel to Destination" invokes the "Plan Route" task model and the "Travel" task model. In other words, "Plan Route" and "Travel" are subordinate to "Travel to Destination". The *Specialization* relationship is denoted by the UML symbol used for this purpose. It is a relationship that links a task model to its super task model. Hence, for example, "Travel by Car" and "Travel by Public Transport" specialize "Travel"; i.e. they are specializations of the generic "Travel" task model.

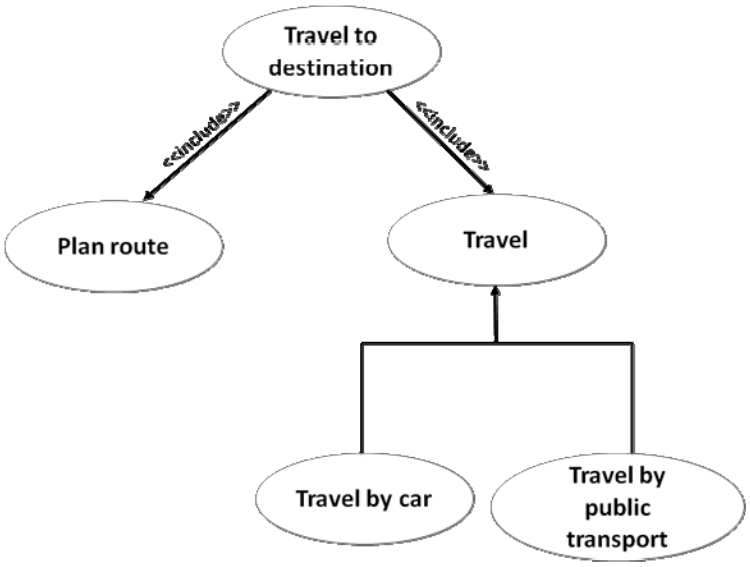

Figure 15: Task-Model Diagram

5.2.1.3 SeCSE's Service Discovery Algorithm

Before outlining the approach, we briefly describe the current SeCSE algorithm for service discovery, called EDDiE, which is used to develop the task-oriented service discovery extension. EDDiE formulates service queries from use case and requirements specifications developed using our UCaRE prototype [43]. This section summarizes the algorithm's description. A full description is provided in [40].

The algorithm has the 4 key components shown in Figure 15; the *Natural Language Processing*, *Word Sense Disambiguation*, *Query Expansion* and the *Matching Engine*. In the first the service query is divided into sentences, then tokenized and part-of-speech tagged and modified to include each term's morphological root (e.g. *driving* to *drive*, and *drivers* to *driver*). Secondly, the algorithm applies procedures to disambiguate each term by defining its correct sense and tagging it with that sense (e.g. defining a *driver* to be a *vehicle* rather than a *type of golf club*). Thirdly, the algorithm expands each term with other terms that have similar meaning according to the tagged sense, to increase the likelihood of a match with a service description (e.g. the term *driver* is synonymous with the term *motorist* which is also then included in the query). In the fourth component the algorithm matches all expanded and sense-tagged query terms to a similar set of terms that describe each candidate service, expressed using the *service description* facet, in the SeCSE service registry. Query matching is in 2 steps: (i) XQuery text-searching functions to discover an initial set of services descriptions that satisfy global search constraints; (ii) traditional vector-space model information retrieval, enhanced with WordNet, to further refine and assess the quality of the candidate service set. This two-step approach overcomes XQuery's limited text-based search capabilities.

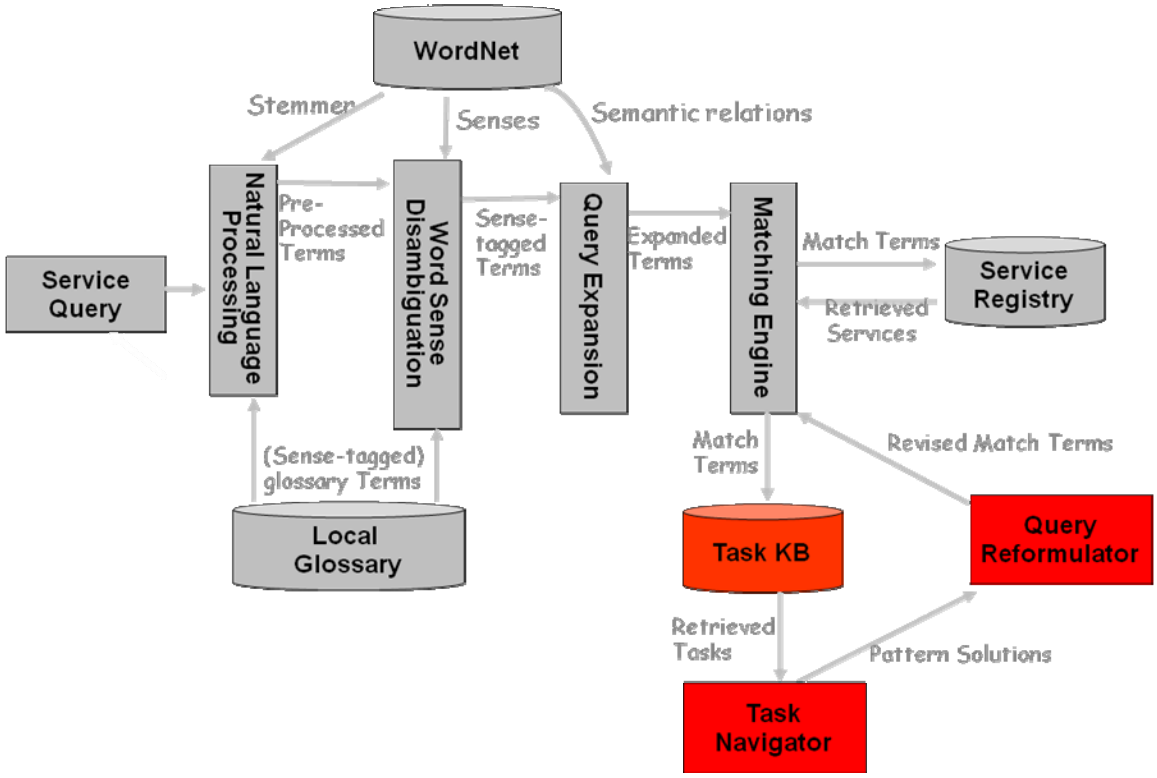

Figure 16: SeCSE's service discovery algorithm enhanced with task knowledge

5.2.1.4 Task-based Extension to Service Discovery

As Figure 15 shows we propose the task-based extension by adding a Task KB and two new components – the task navigator and query reformulator – to EDDiE. Inputs are one or more expanded and disambiguated terms in a service query, and output is one or more new service queries that have been reformulated using retrieved task patterns. Each task pattern in the Task KB includes a structured natural language description of a problem in context, the task structure expressed in CTT, and a structured natural language description of one or more typical service that are proven solutions to the problem. Task-based service discovery is in 5 stages:

1. *Task pattern match*: EDDiE uses the query expansion and word sense disambiguation techniques to match the service query derived from the requirements specification to problem

descriptions of each task pattern in the Task KB. The result is an ordered set of tasks that match to the service query;

2. *User's task selection*: the ordered set of matched tasks is sent to the Task Navigator where the user selects one or more appropriate tasks from the list displayed on Task Navigator. These tasks may be very high level and abstract. If so, the user is asked to choose a node from among the sub-tasks linked to the selected task;
3. *Revise service query*: the Query Reformulator uses all selected (sub-)tasks to generate one or more new service queries that encapsulate both knowledge of the problem domain from the original service query and knowledge from the candidate (sub-)tasks. The latter knowledge includes descriptions of typical services that are likely to be invoked over time;
4. *Service match*: the service discovery algorithm uses each revised service query from step 3 to discover candidate service specifications in service registries. The result is an ordered set of service specifications that match to the revised service query;
5. *Service retrieval*: EDDiE uses the results from step 4 to retrieve specifications from the service registries and compose mappings between attributes and terms in the original service query and each specification. Each specification and its mapping is used by system developers to select or reject each service.

5.2.1.5 Conclusions and Future Work in Task-Based Service Discovery

We are currently building a prototype task knowledge base. In the first stage we are eliciting domain-specific knowledge that describes service-centric solutions for known tasks in a selected automotive domain based on the S-CUBE integrated scenarios. We will then extract the domain-specific task knowledge to generate domain-independent task knowledge that can be reused, similar to the KADS approach to knowledge modeling. Such domain-independent task knowledge can be used to describe more specific task knowledge in a new domain that decreases the cost for expanding the coverage of task knowledge. In turn we will elicit the task knowledge in 3 phases:

1. Discover and elaborate key tasks in S-CUBE scenarios;
2. Model and validate each task model using CTT;
3. Model and validate relationships between the tasks established in the first two phases to generate Modular Task Models (5.2.1) as well as Cooperative CTT Task Models (5.3.1) used to produce the first-cut task knowledge base.

Two important requirements on each task model are that: (i) each task is sufficiently general to be applied across domains and across designs within a domain, and; (ii) the descriptive part of each task model is rich enough to match to service requests using the SeCSE service discovery algorithm.

To achieve these requirements we will use the WordNet online lexicon to produce descriptions of a task models. Once an initial task knowledge base is in place, SeCSE's service discovery and composition algorithms will be extended based on the methods and processes outlined in Section 5.2.1 and 5.3.1. The implementation of the codified knowledge using the SeCSE platform will be tested and validated through future empirical studies.

5.2.2 Codifying User Knowledge during Service Discovery

Our second research direction was to investigate the effect of codified HCI knowledge about users, in the form of user models and profiles, on service discovery. In particular we investigated the effect by incorporating additional knowledge about users from existing models of users and user profiles into service queries implemented in the SeCSE service discovery tools.

There are different definitions and understandings of user models in HCI and other domains. In S-CUBE we assume that user models are,"*models that systems have of users that reside inside a computational environment"* ([44]). Therefore, user models are systems' representations of the properties of a user, such as his or her personal characteristics or preferences. We conjecture that

knowledge about user models be applied to effect the discovery of best-fit services for the user by enriching service discovery queries using the SeCSE platform and refining them based on the preferences and possible constraints expressed in the user model.

In S-CUBE we have not identified previous research applying user models to service discovery. Therefore we sought to use existing service discovery mechanisms to investigate the effect of different user model knowledge on service discovery using the available SeCSE environment. This section of the deliverable reports some examples of initial empirical results to inform future research about codified HCI user model knowledge.

5.2.2.1 Refining Service Discovery Queries with Codified User Knowledge

UCaRE is a module in the SeCSE service discovery environment that permits the generation of service queries from use case and requirements specifications. Each query generated is a structured XML file containing natural language statements, to which we propose to add additional statements or terms extracted from the user model before the query is passed on to the service discovery engine (EDDiE, also in section 5.2.1.). The addition of terms derived from the user model can lead to changes in the discovery and ranking of services for the user. Therefore, we experimented with the inclusion of terms from existing user models in well-established service queries from the SeCSE project to examine the effect on service discovery. At the time of the investigation the SeCSE service discovery environment was linked to a federation of SeCSE service registries that contained over 250 available service specifications drawn from a number of domains. Consider the following example.

A specification of the getRoute use case specification was input into the UCaRE system. The specification described the précis, normal course and associated requirements for the getRoute service of a route mapping system. For example, the use case précis read, "*A driver is driving a car. The driver needs to find the route to his destination. The driver activates the in-car route mapping service. The route mapping service finds a route from the driver's current location to his specified destination*". Likewise, requirements associated with the use case and input into UCaRE read, "*The system shall allow the driver to specify this destination*". In UCaRE, content from this use case specification is selected directly to generate service queries. Because the use case specification was generated using traditional requirements techniques, these queries are unlikely to embody codified HCI knowledge about different service consumers, who are the drivers of the car.

To investigate the effect of codified HCI knowledge we generated 2 different queries for the discovery of best-fit in-car route mapping services for the user U. The queries were called Q1 and Q2.

Q1 is generated using exclusively the information from this use case in the UCaRE system (see Figure 16), so it did not include codified HCI knowledge.

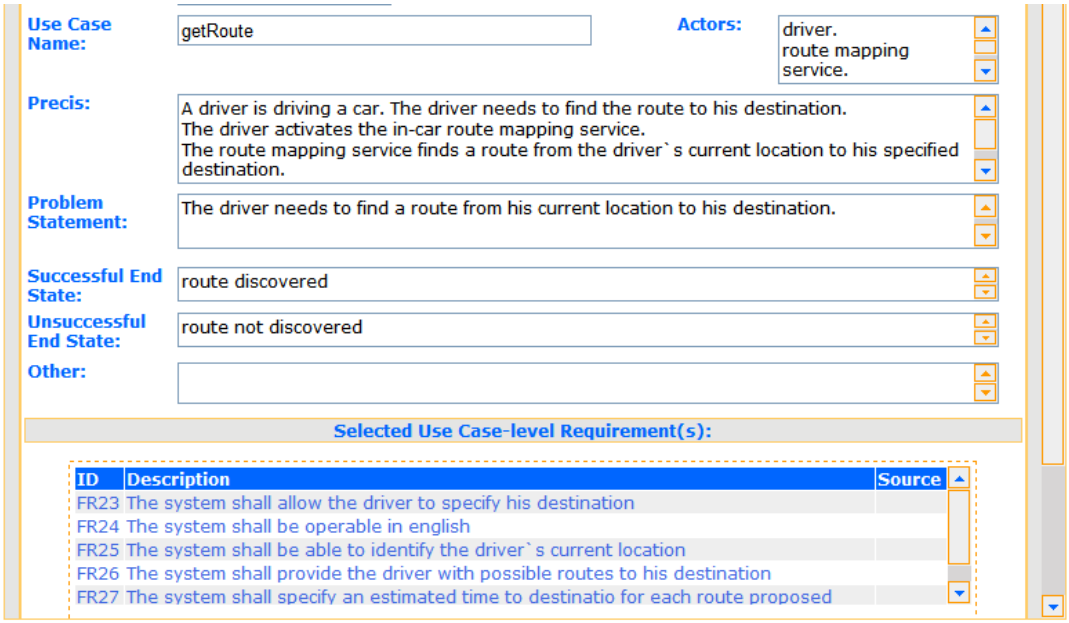

Figure 17: The service query Q1

The services that were discovered were ranked and presented in descending order of fit to the service query, according to the match percentage of the discovered services with the use case requirements (Figure 17); all data pertaining to the generation of Q1 is recorded. Figure 17 shows the discovery of ordered services. Three services were discovered with maximum match scores – *GarminStreetPilot2820GPSNavigation*, *ViaMichelinFindNearbyPOIwebsite* and *MS2000PortableNavigationdevice*.

ServiceName	Description	Match Value			
GarminStreetPilot2820GPSNavigation	The Garmin Street Pilot 2820 systwem comes with preloaded street maps and built in wireless capability which lets its consumers to make handfree calls. It provides real time traffic information; MP3/audio book playback and an extensive Point of Interest database, all packed in one device	100	☑	[Match]	[NFReq]
ViaMichelinFindNearByPOIwebservice	the "FindNearbyPOI" Web Service allows searching for a certain number of addresses or locations closest 'as the crow flies' to a particular address or place of interest within a user-definable search radius. Then displaying any detailed poi information is possible. For example, it can look for car dealers closest to a given location or find competitors that are closest to your sales points and, as a result, analyse catchment areas that are the least well served	100	☑	[Match]	[NFReq]
MS2000PortableNavigationdevice	This device offers a highly user friendly interface for providing voice and video directions and route calculation. Using the GPS system the device calculates its starting position and calculates the route to the destination entered by the user. It contains maps for the whole of Europe	100	☑	[Match]	[NFReq]
Business Trip	The BusinesssTrip service supports users in planning a car trip. It is able to maintain and monitor the schedule for the trip, and also help the driver locate a parking space once they have arrived	97	☑	[Match]	[NFReq]
ServiceManager	Service Manager Module is responsible for the communication between the onboard device and the Xservice portal by means of the Xservice endpoint module	92	☑	[Match]	[NFReq]

Figure 18: Results from the service query Q1

We also generated the second query, Q2 from data from the getRoute use case but with additional information generated from an example user model of user U (Figure 18) based on a simple user model of a car driver. We expressed the user model as informal attribute values expressed using

numbers and natural language expressions. This fitted with the SeCSE service discovery approach, which uses natural language expressions of services and service queries to facilitate discovery.

User model data	
Age	42
Gender	Female
Nationality	French
Address	101 New Street, London N1 1NN
Impairment	No impairments
Driving license issue date	July 2008
Driving record	Clean – no endorsements
Driving experience	Novice
Preferences	Avoid freeway; have break every 2 hours; use voice controls; get turn-by-turn directions

Figure 19: Example user model for a service consumer – a car driver

Therefore we extended the first service query Q1 to generate service query Q2 by adding keywords such as *novice*, *French*, and *turn-by-turn directions* in the use case attributes, as shown in Figure 19. The changes to the use case précis are highlighted in the red circle. Underpinning our approach was identified sensitivities in the EDDiE service discovery algorithm. Experimental and empirical evaluations in the SeCSE project revealed that discovered services and their ranking is dependent on the inclusion and removal of selected key terms and use case attributes from service queries [SeCSE ref].

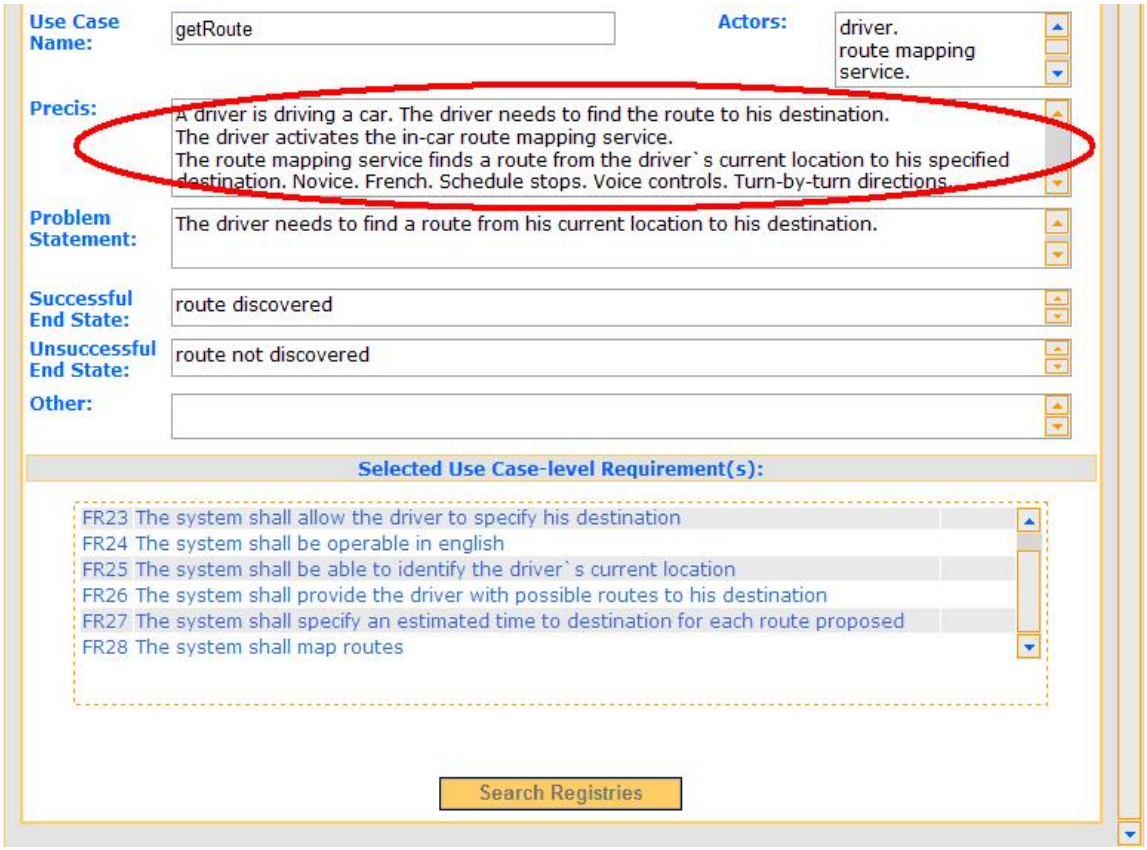

Figure 20: The service query Q2

Figure 20 reveals that the services discovered for query Q2 differ slightly from those found for query Q1. The match value of one of the services, called *Business Trip*, has increased from 97 to 100, confirming it as one of the best-fit services available for user U. Additionally, another service - *Mobile7NavigationKit* - has replaced *ServiceManager* as the fifth best-fitting service for user U.

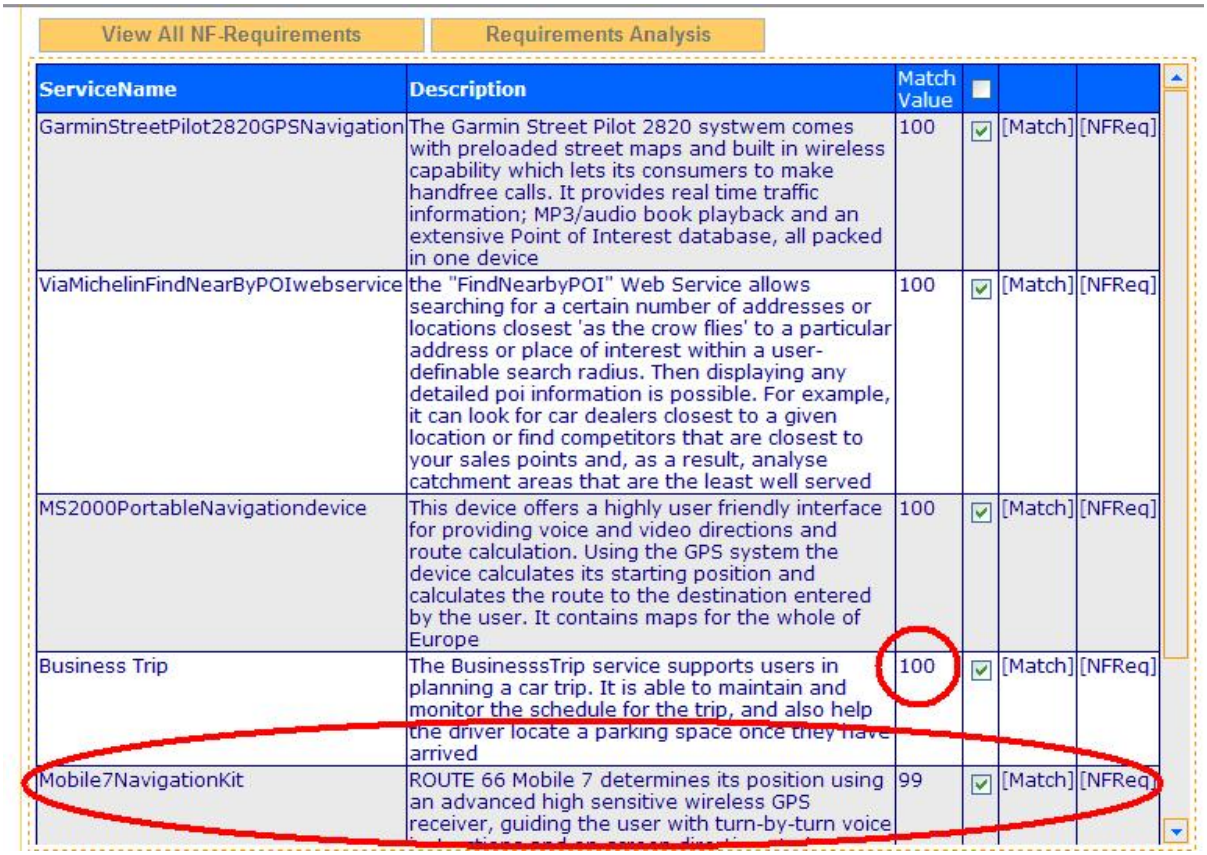

Figure 21: Q2 – Results from the service query Q2

Examining the matched terms from *Mobile7NavigationKit for Q2*, it is apparent that although the information about the user's driving expertise and spoken languages did not influence the discovery of suitable services, his preferences for using voice controls and receiving turn-by-turn instructions did, as is demonstrated in the SeCSE screen shot in Figure 21.

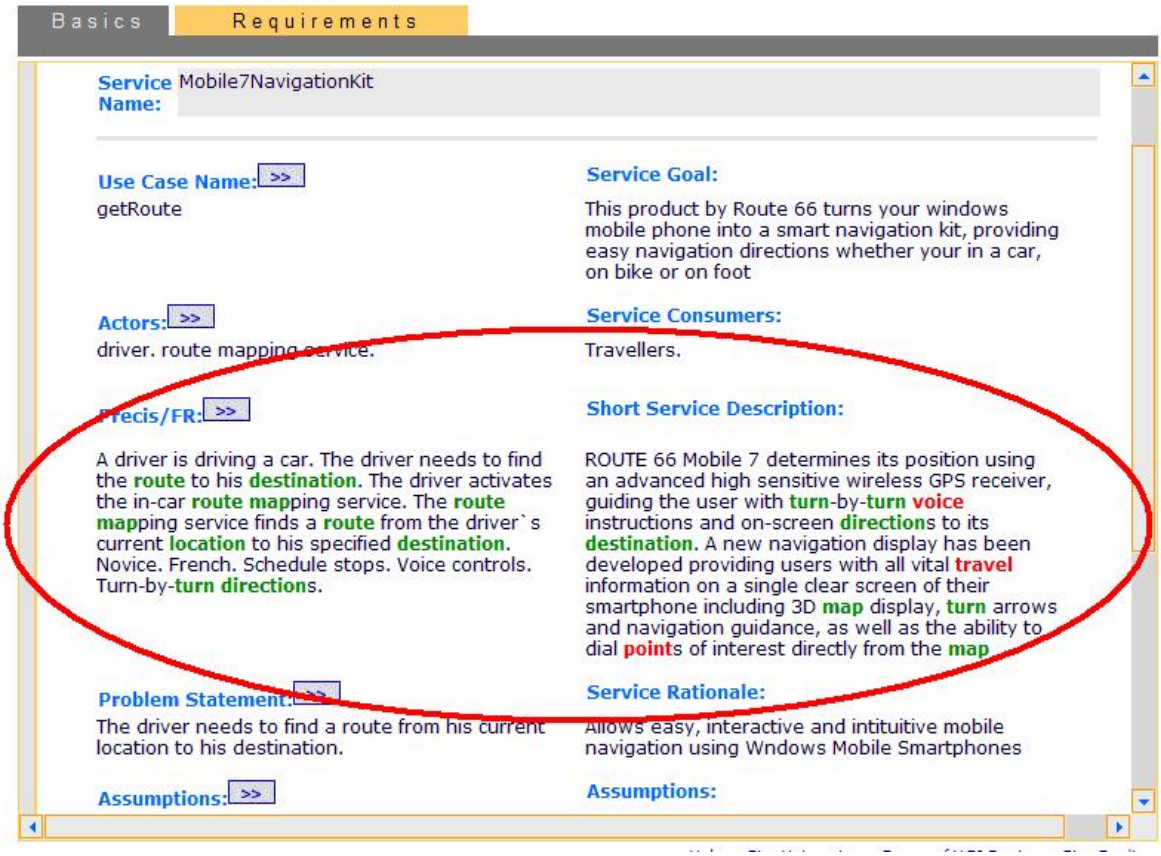

Figure 22: Q2 – matched terms for Mobile7NavigationKit service

5.2.2.2 Verification and Conclusions

The simple example presented in this section demonstrates the impact of including simple user knowledge during service discovery and selection. The example is just that, an example to demonstrate a concept and an effect and suggests that, with suitably robust and sensitive tools such as the EDDiE service discovery tool, knowledge about users can affect the use of service-based applications. The next stages of the work need to map out more systematically how these effects can be achieved. Currently we envisage a three-stage process.

In the first stage we will investigate more systematically, through conceptual and theoretical work underpinned with the integrated S-CUBE scenarios, what user attributes and characteristics are likely to affect the discovery, selection, composition and monitoring of services and service-based applications. The original S-CUBE proposal conceived of such characteristics, for example older users identified by their age attribute think and work using different strategies to younger users, which can be codified into heuristics to be applied during service composition for a service-based application.

In the second stage we will codify this knowledge using precisely defined computational user models that can be accessed in run-time by service-based applications and their run-time environments. In the third stage we will connect these computational user models more formally with existing tools for service discovery, selection, composition and monitoring in the form of heuristics and principles to tailor the operation of these tools for discovery, selection, composition and monitoring. Such connections will allow more empirical investigation of the effect of codified HCI knowledge.

5.2.3 Codifying HCI Knowledge about Organizational Culture to Inform Service Selection

One role for the codification of organizational culture knowledge that we explored in S-CUBE is in service selection during the service discovery process. In this section we report results that demonstrate the codification of organizational culture knowledge during service selection. It is in 2 parts. The first part describes the work undertaken to codify the knowledge. The second demonstrates this knowledge during service selection, in the form of rules that have been developed to support and prioritize services according to quality of service compliance.

5.2.3.1 Codifying Organizational culture knowledge

The meta-model of organizational culture knowledge reported in Section 4.4 and depicted in Figure 9 associates the qualities of business processes and the services invoked in these business processes with different dimensions of organizational culture reported in Section 2.2. We undertook a logical analysis to associate business process qualities to organizational culture dimensions. This was undertaken with a method that combined personas and argumentation structures to associate service qualities with organizational culture dimensions. A persona is a representation of a fictitious individual that embodies the characteristics of a target population. It is constructed using demographic and behavioural user data; they are used in the design process to promote the consideration of users in design decisions. The argumentation structures externalize the knowledge about selection of different qualities. The full method and results are reported in [45].

An example of an argumentation structure developed in S-CUBE is reported in Figure 22. The left hand side of the Figure shows one business process or service quality. The middle part of the figure shows the cultural dimensions and the right hand side associates the arguments to characteristics of personas reasoned about in the method. Each argument can contribute either positively or negatively towards defining a cultural dimension. These cultural dimensions then contribute negatively or positively towards a business process quality, depending on the definition / argument given from the personas. All arguments for or against a specific business process quality are added up and symbolized by a number of plus or minus signs. The more pluses to a dimension, then the more the dimension weighs towards that desired business process quality and vice versa for a minus.

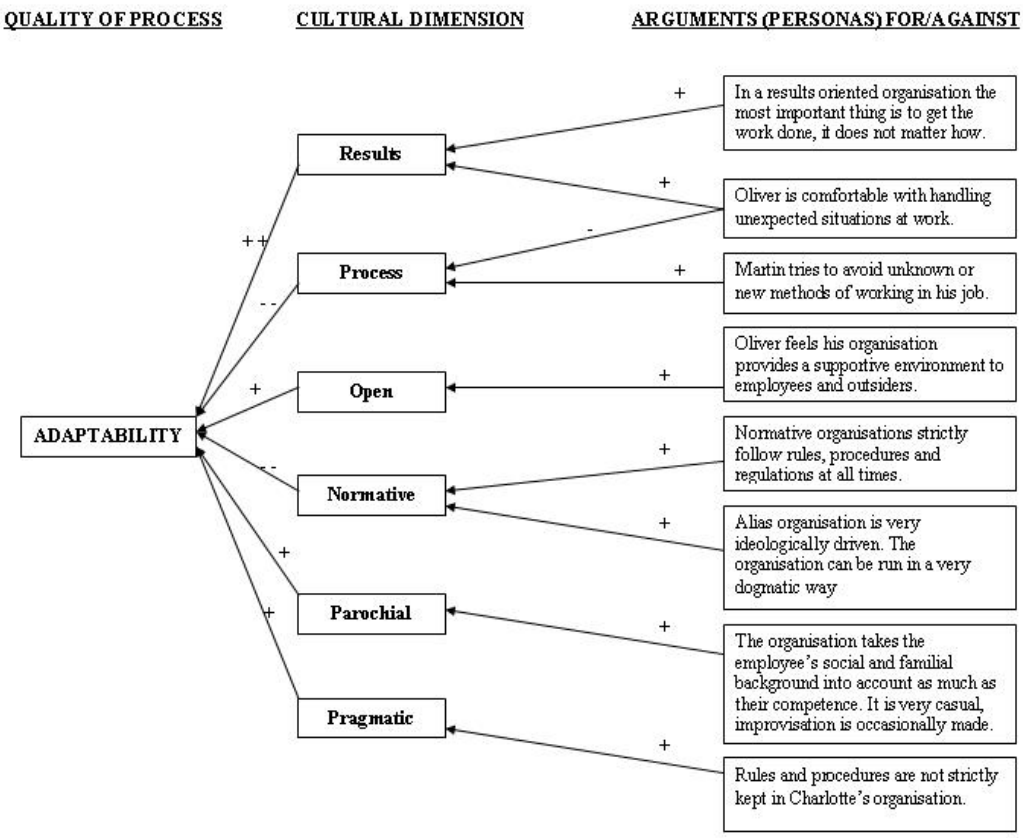

Figure 23: An argumentation structure linking one process quality – availability – to different cultural dimensions identified by Hofstede

We documented the results of the logical analysis in a matrix that associated qualities of business processes and services with organizational culture dimensions. The resulting matrix is shown in Figure 23. The cultural dimensions are listed in the vertical column. Section 2.3 provides the definitions of these dimensions. The business process and service qualities are reported in the horizontal dimension. A '+' in the cell in the matrix indicates that an organization with an organizational culture specified using the dimension listed is more likely to require business processes and qualities with the services indicated. Likewise a '-' in the cell in the matrix indicates that an organization with an organizational culture specified using the dimension listed is less likely to require business processes and qualities with the services indicated.

	Availability	Accuracy	Adaptability	Assurance	Cost	Dependability	Maintainability	Performance	Reliability	Responsiveness	Robustness	Timeliness
Process Oriented	+		- -			-						
Results Oriented			+ +			+	+		+		+	
Parochial dependent			+									
Profesional dependent				+								
Loose control						-						+
Tight control						+						-
Normative organisation		+	- -							-		
Pragmatic organisation			+		+					+		+
Employee oriented					+							
Job oriented					-			+				
Open communication			+	+								
Closed communication		+		-								

+ Cultural Dimension has Positive Weighting Contributing to the Quality of the Process

– Cultural Dimension has Negative Weighting Contributing to the Quality of the Process

Cultural Dimension has Neutral Weighting Contributing to the Quality of the Process

Figure 24: A matrix showing the associations between different culture dimensions and different business process qualities, generated from the argumentation structures as demonstrated in Figure 22. The color of each dimension coincides with the color of each dimension in the design rationale.

Using the matrix we codified the organizational culture knowledge as service selection heuristics, as one example of how codified organizational culture knowledge informs the design and use of service-based applications. We generated these heuristics by associating, with meta-models, dimensions that depict an organizational culture with more important qualities of the business processes in that organization, and hence with the qualities of services that need to be selected and invoked in these business processes. The underlying assumption is that different organizational cultures support business processes that have different qualities, and these qualities are also needed by the invoked services. Therefore, to link the culture dimensions to business process qualities and service qualities, we codified the heuristics to be consistent with the SeCSE Quality of Service Ontology [46]. The Quality of Service Ontology identifies the following measures for the different quality types:

- **Accuracy:** percentage of errors / number of customer problems;
- **Adaptability:** low number of customer problems / low error rate / customer satisfaction;
- **Assurance:** customer satisfaction (percentage metric) / percentage of errors / dependability;
- **Availability:** probability of availability on demand / availability as percentage uptime;
- **Cost:** cost per transaction;
- **Dependability:** high availability / low rate of failure / low mean time to recover;
- **Maintainability:** mean time to recover (time metric) / mean time to repair;
- **Performance:** throughput / mean time to complete / delay / latency;

- **Reliability:** Rate of occurrence of failures / probability of failure on demand / number of failures likely to be experienced per unit of time / security /availability (dependent on maintainability) / mean time between failures;
- **Responsiveness:** mean customer response time;
- **Robustness:** error rate / error handling;
- **Timelessness:** mean resolution time.

The full first set of 13 codified heuristics is reported in [43]. This deliverable includes several examples of these heuristics. Some of these heuristics prescribe minimum levels of service quality that a service must be specified to comply with to be selected. Other heuristics modify the levels of service quality that must be complied with a service as specified already in system requirements, thereby in effect acting as a parameter on the minimum levels of required service quality. Examples of both type of heuristic are reported.

One example of a codified heuristic that define minimum levels of service quality for a service to be selected is:

Heuristic name: Process-oriented-Service-Availability
IF the organization is process-oriented
THEN select all services ≥ 90% uptime [availability quality-of-service]

The heuristic specifies an absolute minimum value associated with the availability of a service. It is applied when the organization culture indicates that the organization seeking to consume a service is process-oriented, and values how work gets done in business processes. Therefore the availability of specified business and services is important, as it strongly influences successful completion of the work. The application of the heuristic ensures that all selected and invoked services have a minimum level of availability.

One example of a codified heuristic that moderates levels of service quality for a service that is specified in the systems requirements are specified below. Consider the following performance requirement that requires a service response time ≥ X seconds:

Heuristic name: Job-oriented-Service-Performance
IF the organization is process-oriented
THEN select all services ≥ X*(0.5) seconds [performance quality-of-service]

The heuristic moderates the already-specified minimum value associated with the performance of a service. It is applied when the organization culture indicates that the organization values the completion of work rather than employee satisfaction, and values getting the work done quickly in business processes. Therefore the performance of specified business and services is important. The application of the heuristic ensures that all selected and invoked services have a moderated, higher level of performance in terms of reduced maximum performance quality-of-service.

5.2.3.2 Verification and Conclusions

Initial results reported in this section are exploratory and preliminary, and describe literature analysis and conceptual modelling work that was undertaken to generate a first set of 13 service selection heuristics. Clearly the heuristics are first versions and unverified in service-based applications. However, the underlying model and its rationale, that identified dimensions of organizational culture can influence the required qualities in the business processes in that organization, and therefore the required qualities of the services that are invoked in these business processes, demonstrates the direction for future work. The next stage for this work is to verify the first set of selection heuristics with external experts. If the heuristics are verified, then we will implement them in SeCSE's service selection tools, with extensions to these tools to enable business analysts to indicate possible values of organizational culture based on established questionnaire techniques. Such an implementation will

enable more empirical exploration of the basic hypothesis – that codified knowledge about organizational culture can inform software service selection.

More generally, this preliminary research was included in the deliverable to explore the potential boundaries of which HCI knowledge to seek to codify in service-based applications. The association between organizational culture and service-based applications is not immediate nor obvious, even after demonstrate. As such, S-CUBE is exploring the codification of organizational culture knowledge as a boundary case for the research in this work package. Future work will continue to explore the boundaries of the research.

5.3 Service Composition

The purpose of a service composition process is to provide all the architectural elements for the conception of the service centric system, as well as the files containing the implementation of the composition. Service composition not only takes into account the system requirements but also the architectural issues and available services. This process consists of several sub-processes that define the main steps and the artifacts produced in order to create a composed service.

In S-CUBE we conjecture that codified HCI knowledge can be used to inform service composition during the architecture design for a service-based application. We plan to explore the proposal through extension of another SeCSE development tool – the Composition Designer. In future work we will extend the Composition Designer to allow a service integrator to generate a service composition with user task models in order to inform more effective service composition. To do this we present an extension of task models that supports service composition based on the task description of services. Our approach is based on the assumption that the behavior of a service can be approximated through its role. In general, within a cooperative task model the execution of a task of one model may enable or disable tasks in other task models. Within our approach we not only distinguish between different roles but also between different services fulfilling the roles.

The behavior of each service role will be modeled using a role task model. At runtime an instance of the corresponding role task model for each active service will be created. The run-time instance of the role task model will capture the enabled task set of a particular service. Cooperation of services will be defined in terms of a global constraint language. The constraints express temporal dependencies between tasks of different services, which in turn will be captured in different instance task models. In essence, we will define a cooperative task model as a tuple consisting of a *set of roles*, a *set of task specifications* (one for each role), a *set of services* where each service belongs to a certain role and a *set of global constraints.*

Next, for each service, an instance of the respective role task model will be created. It is important to note that the approach will be based on the assumption that in limited and well-defined domains the behavioral characteristics of a service matches, more or less, the stereotypical behavior captured in its role-task model. Constraints between tasks of different instance task models will be defined in a language called Task-Constraint Language (TCL)[42]. TCL is not hierarchically structured, which allows us to express constraints between arbitrary tasks of the instance task models. The proposed approach can avoid redundancies and duplications as the structural breakdown of cooperative tasks does not need to be re-specified. Moreover with TCL it is possible to define constraints between multiple instantiations of the same role task model.

The remainder of this section provides more details on the approach.

5.3.1 The TCL Task-Constraint Language

The basic structure of a constraint expressed in TCL is similar to the one of a CTT binary expression. It consists of a left operand, a temporal operator, and a right operand. The operands signify tasks, whereas the temporal operator expresses the type of the constraint. Tasks are identified in two steps: First, we will select the instance task model(s) the task belongs to. Second we will select the task within the model(s). The following is an example constraint expressed in TCL:

PlanRoute.oneInstance.DisplayRoute>>ParkCar.allInstances.ComputeListOfCarParks

The statement can be paraphrased as follows: *After the task DisplayRoute in any task model instance of role PlanRoute is executed the task ComputeListOfCarParks in all task model instances of the role ParkCar becomes enabled.* In other words, only if a PlanRoute service (there may be more than one) has displayed the route to a destination, car park location or booking services (that belong to the ParkCar service role) are enabled to compute the list of available car parks. An examination of the operands (>>) reveals that instances will be identified in two ways. In the case of the left operand, the qualifier "oneInstance" is used to denote that *one* particular instance of the role task model "PlanRoute" is arbitrarily selected. In the case of the right operand, "allInstances" is used to select all existing instances of role "ParkCar".

5.3.2 Design-Time Task-based Extensions to Service Composition

This section outlines the approach that our research is leading us to for task-based extensions to service composition at design-time. The approach is outlined with a simple and established example from the automotive domain.

5.3.2.1 The xTrip Automotive Example

In the example an automotive manufacturer is looking to use web services for different purposes. One is to improve customer satisfaction by providing the car with a new service that allows the car owners to remotely manage their trips, guaranteeing that new trips will not overlap already existing appointments and if such overlap happens this could be solved with a call to a car owner's choice number in order to try to arrange a new appointment. This new service is called *xTrip*.

The *xTrip* service calculates the route between two places indicated by the user or between the current car position (GPS coordinates) and the destination address indicated by the user. The system uses this information and an external Geographical Information System (GIS) to estimate the route and time needed to arrive to the destination and select the quickest route between alternatives. The service then consults the user agenda to determine if there are conflicts with existing appointments. If there are one or various routes without conflicts, *xTrip* will include in the agenda the trip, showing that the driver will be busy during the time he is travelling. If there is an overlap and there is no alternative to solve it, the service will call a phone number indicated by the user to solve the problem. It is supposed that the user will be able to postpone or cancel an appointment. The selection of the service to perform this call depends on the telecom provider that offers the best rate to connect the two endpoints and also the best performance.

Five sub-processes (three that are extended task-related information) will constitute the service centric architecture and composition design process. These 5 sub-processes are reported in the next 5 sub-sections.

5.3.2.2 Enterprise Architecture Engineering

The purpose of the Enterprise Architecture Engineering sub-process will define and maintain domain-wide or organisation-wide architecture models and assets with their benefits and drawbacks. In this process, no task-related extensions are realised.

5.3.2.3 Service Specification Architecture

The purpose of the Service specification Architecture sub-process will be to analyse what kind of services (abstract services) will be needed to cover all the functionality of the system. At this stage, the service specification architecture acts as the 'placeholder' architecture. It defines the optimal abstract services descriptions aligned to the business requirements and fulfils the needs of a strategic direction for architecture. From the system requirements, three abstract services are identified to: (i) calculate the route between two places; (ii) manage and control the agenda; (iii) perform a phone call. The services are called *CalculateRoute*, *ManageAgenda* and *PerformPhoneCall* and are part of a possible service composition. The behavior of each abstract service will also be matched to role task models

contained in the Task KB[2]. In the example the abstract service *CalculateRoute* is matched to the role task model called *PlanRoute* that plans and calculates the best route between two places.

5.3.2.4 Design Time Service Composition

The purpose of the Design Time Service Composition sub-process will be to define at design time an appropriate composition of service specifications fulfilling both functional requirements, in the form of the workflow to be followed, and non-functional requirements (in the form of requirement types such as security and performance) defined in the previous sub-processes. It will therefore specify the behaviour of the system and how the functionalities will be choreographed or orchestrated. All the information is presented with models, so it is an abstract representation of the composition. In our example, the composed service gets a route for a planned trip and then, checks the compliance between that trip (in terms of expected length) and the appointments scheduled on the user agenda. In case of conflicts, the service calls automatically a specified phone number to try to solve the conflict. Here, task constraints for any matched role task model will be fed into this process to inform service composition, e.g. PlanRoute.oneInstance.DisplayRoute>> ManageAgenda.oneInstance.CheckTripComplianceWithAgenda.

5.3.2.5 Composition Realisation Architecture

After identifying the abstract services and designing the service composition flow, a list of available candidates will be defined. Descriptions and specifications of abstract services that cannot be matched to specific role task models in the second sub-process will be used as before to find concrete services that can fulfill the requirements. In our example services (e.g. *CalculateRoute*) that were matched to specific role task models (e.g. *PlanRoute*), are used to discover concrete services that belong to specific roles (e.g. *GetRoute, xNavigation*). In this process candidate services and possible alternatives will be selected using the role task models, i.e. for each service an instance of the respective role task model is created. In our example, the services that best matched the composition needs are selected; being these services the *xNavigation* (to calculate the route) and *xAgenda* (to control the agenda). So, these services cover the functionalities required. When the design is finished the composition flow will be translated into an executable script, using language such as BPEL or the SeCSE composition language.

5.3.2.6 Validation

The purpose of the Validation sub-process will be to guarantee that selected web services will meet their specified functionalities and qualities. It will validate the design, the availability of the system, the ability to build the design correctly, the ability to reproduce the system, the ability to correct faults, etc. In this process, no task-related extensions will be realised.

5.3.3 Run-Time Task-based Extensions to Service Composition

At run-time, the binding and re-binding process will be launched when a fault is detected during the invocation of a service or if QoS constraints will be violated. If a service is not available, the re-binding process will try to look for an equivalent service and do a re-bind to change the service with a similar one, allowing the system to proceed with a normal execution of the composition. At this stage role task models will enable the process to replace a deficient service with an equivalent service that belongs to the same role and have similar task models.

5.3.4 Verification and Conclusions

The above processes remain to be implemented and validated. However, again, the development of the processes using codified HCI task knowledge in the form of cooperative CTT models indicates that there is a potential impact of user task models on service composition. The extent of that impact will be explored through future conceptual and empirical research in work package JRA1.1.

[2] The Task Knowledge Base (Task KB) stores amongst others role task models that have been created previously.

5.4 Service Monitoring

Service monitoring is concerned with the observation of services' behavior at run-time to ensure their adherence to constraints and expectations that are usually defined in requirements and expressed in service-level agreements. In this section we report the results from preliminary research to explore the use of codified HCI knowledge, this time about user experiences, to monitor services.

A user experience (UX) is defined as the quality of the overall experience that a user experiences when interacting with a service. Elements of the user experience may include implementation-specific aspects of services (e.g. the visual design of the user interface if any) but also, importantly, properties of the service related to the execution of its functions. For example, a holiday booking service that would require the booking of accommodation before that of flights can be considered to illustrate the latter point – the flow of the processes executed would be less than optimal for any user of the system. Therefore we consider that the *user experience* is a property to monitor that is associated with other service-related properties more traditionally monitored with service-centric systems. The preliminary research used simple examples of current UX techniques to explore codified HCI knowledge in the form of rules and patterns that can be developed in S-CUBE.

5.4.1 Monitoring User Experiences and Services: A First Example

The scenario considered was that of a user looking to use a service running at or above a specified acceptable level of UX. The scenario explores the challenge of finding suitable UX metrics with which to create and define suitable monitoring rules. The next 2 sub-sections report preliminary results in this direction.

5.4.1.1 User Experience Metrics

A quantitative rather than qualitative metric was sought to be used as a threshold in UX monitoring, and the approach developed by Joshi et al. ([47]) was adopted for the calculation of such a metric. As described in [47], the User Experience Metric (UXM) spans a scale of 0 (the worst possible UX rating) to a 100, and requires the specification of:

- High level user experience *goals* that have been deemed relevant to the service assessed, each of which can be decomposed into parameters contributing to the achievement of the goal and its measurement
- *Weightage* between 0 (least important) to 5 assigned to goals and parameters, this to indicate their relative importance and the prioritisation attributed to them by the service stakeholder
- *Scores* between 0 (the lowest possible) to 100 indicating the estimated level of achievement of the goals and parameters

Guidelines must also be made available to help the interpretation of goals and parameters for a particular service, and to support the attribution of weightages and scores that are reflective of the assessor's opinion.

Once these have been specified, it is possible to compute the UXM for the service as the sum of the weighted average of the scores of all goals: $UXM = \sum(Wg \times Sg / \sum Wg)$, with $Sg = \sum(Wp \times Sp/\sum Wp)$. Wg and Wp are the weightages of a goal and a parameter respectively, and Sg and Sp are the scores of a goal and a parameter respectively (see Figure 24 for an example of UXM calculation).

Goals	Weightage	Score
Learnability	4	78.6
Speed of use	2	77.1
Ease of use	3	65.6
Error free use	3	67.5
Retention	1	75.0
Subjective satisfaction	2	78.6
UXM Value		73.3

Goal Parameters	Weightage	Score
Learnability		78.6
Conceptual model clarity	3	75
Language understandability	0	0
Minimal learning time	5	75
Consistency with earlier version	1	50
Visibility of choices and data	4	100
Consistency with other products	1	50
Speed of use		77.1
User control and freedom	3	75
No memory and cognitive load	4	100
Internal consistency	4	75
Customization	0	0
Automation and shortcuts	1	0
Ease of use		65.6
Minimal user task load	5	75
Automation of routine tasks	3	50
Error free use		67.5
Good feedback	4	75
Error tolerance	3	50
Error recovery	3	75
Retention		75.0
Retention	3	75
Subjective satisfaction		78.6
Visceral appeal	2	75
Behavioural appeal	4	75
Reflective appeal	1	100

Figure 25:UXM calculation table [47]

This simple example demonstrates that monitoring the UXM requires new monitoring rules. We consider these monitoring rules as new codified HCI knowledge. The next section outlines one possible approach using the *SECMOL* service monitoring language developed within the SeCSE project.

5.4.1.2 New Monitoring Rules in SECMOL

SECMOL was developed in the context of the layered view of monitoring described in [48]. This view categorizes monitoring components into 3 different, interconnected types: *Data Collectors*, *Data Analyzers* and *Recovery Handlers* (see also Figure 25). The data collectors gather monitoring data which they provide to data analyzers, either on request or as soon as the data becomes available. The data analyzers evaluate the monitored properties on execution, and the recovery handlers execute repair processes if a problem is found with the monitored process.

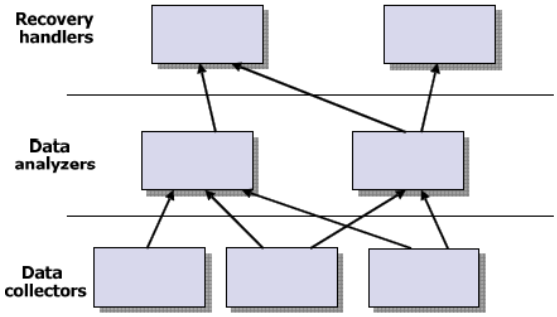

Figure 26: Layered view of monitoring [48]

As detailed in [48], the layered view of monitoring supports 2 monitoring approaches (strict monitoring and loose monitoring), each using a different language (WSCOL and EC-assertion respectively). SECMOL was introduced as a common monitoring language for the specification of monitoring rules. As shown in Figure 26, SECMOL makes use of the following constructs to define monitoring rules ([49]):

- *Runtime:* specifies the endpoint of the service that will perform processing necessary for monitoring (e.g. data extraction);
- *Schedule:* defines time intervals at which some action ought to be performed;
- *message identification*: defines the messages exchanged between services; they are passed to data collectors to inform which messages should be captured at runtime for rule checking;
- *data extraction*: defines the extraction and packaging of data from messages;
- *Computation:* describe a computation over one or more datastreams extracted from messages;
- *Rule*: defines the condition that specifies a service guarantee term.

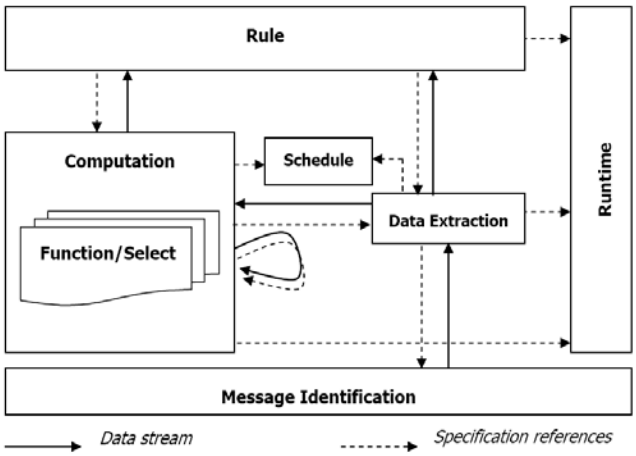

Figure 27: Basic constructs of SECMOL [49]

In S-CUBE we envisage that monitoring patterns can be defined using SECMOL for the specification of typical service properties to be checked at run-time. A monitoring pattern specifies at a high level of abstraction the logic and the computations that need to be performed for a monitoring rule that would be instantiated using the pattern [50]; they can be used as templates that are customisable by users for the development of monitoring rules. The general scheme for developing a monitoring pattern is reported in [50] and requires the specification of:

- *Pattern name;*
- *Keywords* that may be used to retrieve and instantiate the pattern;
- *Granularity* to specify whether the pattern specifies a property of the service or that of a service operation to be monitored;
- *User defined parameters* for the creation of pattern instances (including a description of the parameter role and the valid values for it);
- *Abstract specification of the properties* that may be monitored by the pattern.

The base scenario of a user looking to use a service running at or above a specified acceptable level of UX could then entail that a service provider calculates and specifies a base UXM for his service. Service consumers could provide feedback on their experience using the service by supplying the necessary data for the calculation or re-calculation of the UXM, which would then be used for updating the service's advertised UXM. For each user, the service would be monitored at runtime to check that the UXM does not fall below the threshold they specified.

A corresponding UX monitoring pattern, developed in keeping with the scheme is shown in Figure 27. We introduced *pattern name*, *keywords*, *scope*, *granularity, user defined parameters* and *abstract specification* into the general scheme for developing monitoring patterns. The *SECMOL element* refers to the basic constructs of SECMOL presented in Figure 26. To instantiate a monitoring rule from the pattern, a user would have to indicate the desired *user parameters*. In the *abstract specification*, the *CheckServiceOperationUX* rule is specified; it monitors whether the UXM of the specified service operation falls below the threshold stipulated for the instantiated rule.

Pattern name	ServiceOperationUXPattern				
Keywords	UX, service, operation				
Granularity	Operation				
User defined parameters	**Name**	**SECMOL element**	**Meaning**	**Type**	**Valid value**
	_dataExtractrionRuntimeURL	<Runtime>	Machine where to extract the data from at runtime	xs:anyURI	any
	_monitorRuntimeURL	<Runtime>	Machine where the monitor will reside	xs:anyURI	any
	_service	<MessageIdentification>	Service that provides the operation to be monitored	xs:string	any
	_serviceOperation	<MessageIdentification>	Operation whose usability must be monitored	xs:string	any

	_minUXM	<Rule>	Minimum acceptable usability level specified by the user	xs:float	0...100
	_monitoringSchedule	<Schedule>	Length of time interval (in seconds) at which the usability monitoring rule will be checked	xs:integer	any

Abstract specification of the properties that can be monitored

```
<SLA xmlns="http:[anAddress]" name="ServiceOperationUXPattern">

   <Runtime name="DataExtractorRuntime">
      <RuntimeURL>_dataExtractionRuntimeURL</RuntimeURL>
   </Runtime>

   <Runtime name="MonitorRuntime">
      <RuntimeURL>_monitorRuntimeURL</RuntimeURL>
   </Runtime>

   <Schedule name="UXTimer">
      <Interval>
          <Seconds>_monitoringSchedule</Seconds>
      </Interval>
   </Schedule>

   <MessageIdentification name="ServiceMessage">
      <OperationName>_serviceOperation </OperationName>
      <WSDL>_service </WSDL>
   </MessageIdentification>

   <DataExtraction name="DataExtractionResponse">
         <Runtime>DataExtractorRuntime </Runtime>
         <ReactOn type="Response">ServiceMessage</ReactOn>
               <Package>
                    <ID name="id"/>
                    <Time name="t"/>
                    <Data name="uxm"/>
               </Package>
   </DataExtraction>

<Computation name="CalculateUXM">
     <Runtime>MonitorRuntime </Runtime>
      <ReactOn type="Timer">UXTimer</ReactOn>
      <Input>
                <Stream name="UXMs">DataExtractionResponse</Stream>
            </Input>
            <Function name="averageUXM" location="self">
                 <Argument>
                      <StreamField new="false"
label="id">UXMs</StreamField>
                 </Argument>
                 <ResultStream>
                         <StreamField new="true" name="averageUXM"
type="float"/>
                 </ResultStream>
            </Function>
</Computation>

      <Rule name="CheckServiceOperationUX">
         <Runtime>MonitorRuntime </Runtime>
         <QuantifiedStream>
             <Quantifier>forAll</Quantifier>
             <Stream name="AvUXM">CalculateUXM</Stream>
         </QuantifiedStream>
         <Head>
            <Expression>
            <RelationalExpression>
                <GreaterThanOrEqualTo>
                      <Value1>
                           <StreamField>AvUXM</StreamField
                      </Value1>
                      <Value2>
                           <DirectValue type="Float">_minUXM</DirectValue>
                     </Value2>
                <</ GreaterThanOrEqualTo>
             </RelationalExpression>
             </Expression>
         </Head>
    </Rule>
</SLA>
```

Figure 28: An Example Service Monitoring Pattern

5.4.1.3 Verification and Conclusions

This section reports a first exploration of the use of *user experience* knowledge for service monitoring. It sought to investigate how S-CUBE can embed service monitoring techniques into the wider user experience approach. This user experience approach embodies both explicit, articulated requirements of the service consumer and more implicit, unarticulated requirements that nonetheless inform the required user experience. The concept of the user experience is increasingly important in interaction design, and the design of future interactive service-based applications is likely to be influenced by it. This section has demonstrated, superficially, how user experience can be linked to service monitoring, but more work is needed in this direction to demonstrate the link more effectively.

6 Future Directions to Codify HCI and Context Knowledge in Service-Based Applications

This final section of the deliverable is in 2 parts. The first summarizes the work undertaken and reported in Sections 4 and 5. The second reviews the work undertaken, and uses it to suggest future research directions for codifying HCI knowledge for use in service-based applications.

6.1 Codified Context and HCI Knowledge

Sections 4 and 5 report research undertaken between months 7 and 10 in work package JRA1.1. Section 4 reported conceptual modeling work that identified important context and HCI concepts, and modeled them by associating them precisely to each other and to established software service concepts. This work scoped some of the codified context and HCI knowledge that has been identified. Section 5 applied some of the concepts identified and modeled in Section 4 to different activities important in service-based application – specifying, discovering, composing and monitoring services, and preliminary results are reported in the section. The SeCSE platform was chosen for the implementation of codified HCI knowledge. Although exploratory, the results do indicate important research directions for codifying HCI knowledge in S-CUBE that we elaborate in the next section.

6.2 Research Directions Over the Next 12 Months

One purpose of this deliverable is to inform future research directions in JRA1.1. We have limited resources available to codify context and HCI knowledge, so some prioritization of directions needs to happen to ensure that the next research stage will deliver useful results. To this end the results from section 5, as well as other preliminary research not reported in the deliverable, were used to undertake a review of research potential. The results of this review are reported in the matrix in Figure 28.

The matrix provides a framework for scoring the potential applicability and utility of different types of codified HCI knowledge to 5 different activities in the development and use of service-based applications:

1. Specification and publication of services;
2. Discovery and selection of services
3. Composition of services during architectural and detailed design of service-based applications;
4. Monitoring services in a service-based application;
5. Adapting the service-based application in light of results of monitoring the application.

Against each of these 5 activities we reviewed and scored different types of codified HCI knowledge, some of which have been demonstrated in the deliverable. Figure 28 summarizes the results. The *** show the activities in which we conjecture the application of codified HCI knowledge can make a contribution of higher value, whereas the * show the activities in which we conjecture the application of codified HCI knowledge can make some contribution but of lower value that the activities marked with ***. The empty cells denote areas where no significant possible application of codified HCI knowledge has been identified at the time of producing the deliverable.

Types of codified HCI knowledge	Service specification and	Service discovery and selection	Service composition	Service monitoring	Adapting service-based applications

	publication				
Organizational culture	*	***		***	*
Accessibility standards	***	***	*		
User experience	*	*		***	*
User error modeling		***	***	***	
End-user personalization		*	*	***	*
End-user customization	*	*	*		
User modeling		*		***	***
User task modeling		***	***		*

Figure 29: A matrix showing the potential application to different activities in the development and use of service-based applications of different types of codified HCI knowledge

Each of these types of codified HCI knowledge is briefly reviewed in turn.

6.2.1 Codified Knowledge about Organizational Culture

Conceptual analysis work reported in section 5 supported the knowledge modeling reported in section 4 and identified associations between dimensions of organizational culture, qualities of business processes in these cultures, and the qualities of services that are invoked in these business processes. Therefore, according to results reported here, we might expect codified organizational culture knowledge to be more useful in activities that exploit service qualities more directly. This deliverable demonstrated one potential use in the form of service selection heuristics, but there are also other activities, in particular during service monitoring, where similar heuristics can moderate and change service monitoring rules for different organizational cultures. There is also a potential use of codified organizational culture knowledge during the adaptation of service-based applications. Organizations with different cultures will be more or less amenable to process and service change, hence knowledge of the identified culture dimensions can inform the selection of adaptation strategies that are more likely to be adopted in different organizations.

6.2.2 Codified Knowledge about Accessibility Standards

Some simple development of a new SeCSE service specification facet was reported in section 5. This result demonstrated that software services, like other software artifacts, have the potential to be extended to include standards-based information about accessibility. Therefore knowledge of such standards informs what we need to specify and publish about services in service registries and other

locations. Likewise, if knowledge about compliance of each service with standards such as accessibility is published, then this knowledge can be used during service discovery and selection. Processes at design-time and at run-time can be moderated to elicit directly standards with which compliance might be important, prior to discovering and selecting services.

6.2.3 Codified Knowledge about User Experience

Section 5 reports the results of preliminary research to explore how user experience models and service monitoring might be linked. Clearly the results are tentative, and further research is needed to explore these links further. Therefore, whilst more work is needed, the increasing importance of considering the wider user experience of service consumers using service-based applications will have impacts on technical testing evaluation, whether or not the application is service-based. It means that new techniques for service monitoring should at least be considered in the wider context of user experience methods. As such we have not explicitly codified knowledge about user experiences and user experience methods. Rather, there should be more consideration of user experience techniques during the development of methods to develop service-based applications. We consider this future research direction to be important to S-CUBE.

At the level of method and technique integration, we also consider that user experience techniques can inform and improve service specification and publishing, service discovery and selection, and adaptation of service-based applications, although the potential impact on these activities will be less than on other activities.

6.2.4 Codified Knowledge about User Error Modelling

The human-computer interaction discipline has undertaken substantial research into user errors, and has produced numerous user error models and taxonomies that are available more widely to other researchers. Previous research into user error modeling was described in the PO JRA1.1.1 deliverable. The subsequent review of user error modeling is currently ongoing and not reported in this deliverable, but we believe that there are substantial advantages of user error models to different activities related to service-based applications.

User error models and taxonomies can be used to describe and explain previous user behaviour and predict future user behaviour related to different types of error. If accurate user error models for different types of service consumers in different domains can be developed, then the codified knowledge about user behaviour expressed in these models can be exploited in service-based applications, in particular during service discovery and selection, during service composition, and during service monitoring. If a service consumer, according to the model, is more likely to make user errors during use of one or more services in an application, then the development and run-time environments should discover and select services with functionality and features that can more effectively handle these errors. Likewise, if a service consumer, according to the model, is more likely to make user errors during use of one or more services in an application, then the composed service application needs to be designed to be dependable with respect to these user errors. And, during service monitoring, monitoring rules will need to be extended to monitor for user error and to allow for the effect of possible user error on service performance, reliability and other qualities that can be associated with occurrences of user errors.

6.2.5 Codifying Knowledge about End-User Personalization and Customization

Customization and personalization of devices and applications by individual service consumers is an increased trend, and one that is in theory supported by the adaptability that is delivered in service-based applications. Again work to explore and codify this knowledge is ongoing, but Figure 28 identifies our initial understanding of how the different service-related activities might be impacted by knowledge about end-user personalization and customization. We will report more complete results in this area in future deliverables.

6.2.6 Codifying Knowledge about User Modelling

Sections 4 and 5 of this deliverable described the association between user models, software services and service-based applications, and demonstrated one simple use of knowledge from one informally-expressed user model and its impact on service discovery using an existing service discovery environment. Although the effect of this user model knowledge was small, the result needed to be put into a wider context. The available service specifications were not altered at all to document knowledge about potential end-users. No extensions to the service discovery engine were made to use the user model knowledge. And no review of the improvement or otherwise of the discovered services was undertaken. Clearly, to progress research in this area, more formal work to codify user modelling knowledge and design the use of this knowledge in the different activities is needed, and this codification and design work will be one next step in S-CUBE.

Figure 28 shows that we envisage some impact of codified user model knowledge on service discovery and selection, but potentially much greater impact on service monitoring and adaptation of service-based systems. User models maintain system-based computational models of a user's goals, states and characteristics. Current research suggests that these goals, states and characteristics are more likely to inform service monitoring because each provides important user knowledge with which to inform monitoring rules. There is a parallel here between service monitoring and tool-based diagnosis of user states and errors in environments such as intelligent tutoring systems. Computational models are finer-grain and more up-to date with respect to each service consumer, and hence can provide more relevant information with which to generate and adapt rules for service monitoring. Likewise these more finer-grain and up-to-date models can inform strategies for adapting the service-based applications based on the results of monitoring that ensure that user goals are satisfied. The next deliverable will report substantial research results in this direction.

6.2.7 Codifying Knowledge about User Task Modelling

Sections 4 and 5 of this deliverable also described the association between user task models, software services and service-based applications, and demonstrated the use of the CTT task modelling formalism to represent knowledge about user tasks and introduce this knowledge to inform service discovery, selection and composition. Potentially user tasks models that codify important HCI knowledge have the potential to fill a gap in current approaches for modelling service compositions. Most existing business process and work flow modelling techniques model coarse-grain processes with little support for finer-grain user tasks of different types and interactions with the service-based applications. User task models from HCI naturally plug this gap, and introduce new concepts such as task goals from the user perspective not modelled using approaches such as BPEL. In the deliverable we have already elaborated detailed approaches for extending service discovery and composition techniques with formalised user task models for tasks in the S-CUBE integration scenarios. The next deliverable will report the results from substantial development and evaluation research in this direction. We will also investigate the potential of user task models to inform and moderate strategies for adapting service-based applications.

6.3 Codifying Context and HCI Knowledge: Conclusions

Returning to the S-CUBE Description of Work, we sought to undertake the following research to codify context and HCI knowledge relevant to service based applications engineering:

1. Review related research literature and select formal task and user models with properties that represent codified knowledge about context factors associated with task and user characteristics pertinent to service based applications;
2. Review research into personalized user interfaces and multi-modal interaction to determine rules, patterns and guidelines for system and service-led configuration versus user-led customization of service based applications. In the future we will use a faceted classification scheme of context factors that can be applied to both consumer task and user

models and extended specifications of services, thus providing a common underlying ontology of both services and their contexts.

As the deliverable shows, research in S-CUBE has undertake more work to address the proposal outlined in the first bullet, and has broadened the review of other types of HCI knowledge from accessibility and organizational culture research. The next stages will continue the research described and prioritized in this chapter.

6.4 References

1. **Andrikopoulos, V., et al., *State of the art report on software engineering design knowledge and Survey of HCI and contextual Knowledge*. 2008, S-Cube Project Deliverable.**
2. **Weiser, M., *Some Computer Science Issues in Ubiquitous Computing*. Communications of the ACM, 1993. 36(7): p. 75-84.**
3. **Schilit, B.N., N. Adams, and R. Want, *Context-Aware Computing Applications*, in *1st Workshop on Mobile Computing Systems and Applications*. 1994, IEEE Computer Society Press: Santa Cruz, CA, USA. p. 85-90.**
4. ***CoDAMoS (Context-Driven Adaptation of Mobile Services) - http://www.cs.kuleuven.be/~distrinet/projects/CoDAMoS/*.**
5. **Chen, G. and D. Kotz, *A Survey of Context-Aware Mobile Computing Research*. 2000, Dartmouth College.**
6. **Ghadiri, N., et al., *A Context-Aware Service Discovery Framework Based on Human Needs Model*, in *5th International Conference on Service-Oriented Computing (ICSOC 2007)*. 2007, Springer: Vienna, Austria. p. 404-409.**
7. **Preuveneers, D., et al., *Towards an Extensible Context Ontology for Ambient Intelligence*, in *2nd European Symposium on Ambient Intelligence (EUSAI2004)*. 2004, Springer: Eindhoven, The Netherlands. p. 148-159.**
8. **Mylopoulos, J., et al., *Telos: Representing Knowledge about Information Systems*. ACM Transactions on Information Systems, 1990. 8(4): p. 325-362.**
9. **Pohl, K., *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. 1 ed. 2007: dpunkt.verlag.**
10. **Baresi, L., E.D. Nitto, and C. Ghezzi, *Toward Open-World Software: Issue and Challenges*. IEEE Computer, 2006. 39(10): p. 36-43.**
11. **Kluckhohn, C., A.L. Kroeber, and A. Louis, *Culture: A Critical Review of Concepts and Definitions*. 1952: New York, Vintage Books.**
12. **Hofstede, G., *Cultures and Organisations: Software of the Mind*. 1994, London: Harper-Collins Business.**
13. **Choong, Y.Y., *Cross-Cultural Issues in Human-Computer Interaction*. International Encyclopedia Of Ergonomics And Human Factors, 2006.**
14. **Galdo, E. and J. Nielsen, *International User Interfaces*. John Wiely & Sons, 1996.**
15. **Trompenaars, F., *Riding the Waves of Culture: Understanding Cultural Diversity in Business*. London: Nicholas Brealey, 1993.**
16. **Hofstede, G., *Cultural Dimensions in People Management – The Socialization Perspective*. Globalizing Management–Creating and Leading the Competitive Organization, Wiley & Sons, New York, NY, 1992.**
17. **Hofstede, G. and G. Hofstede, *Cultures and Organizations: Software of the Mind*. 2005, New York: McGraw Hill.**
18. **Krumbholz, M. and N. Maiden, *The implementation of enterprise resource planning packages in different organisational and national cultures*. Information Systems, 2001. 26(3): p. 185-204.**
19. **Sears, A. and J.A. Jacko, eds. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. 2nd ed. 2002, Lawrence Erlbaum: New York.**
20. **W3C, *Web Services Internationalization Usage Scenarios - http://www.w3.org/TR/2004/NOTE-ws-i18n-scenarios-20040730/*. 2004.**
21. **W3C, *Internationalization Web Service Task Force - http://www.w3.org/International/ws/***
22. **Schwartz, J., *Jonathan's blog - http://blogs.sun.com/jonathan/entry/participation_age_cont_d*. 2005.**
23. **W3C, *Web Services Internationalization (WS-I18N) Working Draft - http://www.w3.org/TR/2008/WD-ws-i18n-20080415/*. 2008.**
24. **SeCSE.**
25. **OMG, *UML 2.0 Superstructure Specification*. 2002, Object Management Group.**

26. **Harmsen, F., S. Brinkkemper, and J.L.H. Oei, *Situational Method Engineering for Informational System Project Approaches,* in *Working Conference on Methods and Associated Tools for the Information Systems Life Cycle,September*. 1994, Elsevier Science Publishers: Maastricht, The Netherlands. p. 169-194.**

27. **Wand, Y. and R. Weber, *On the Ontological Expressiveness of Information Systems Analysis and Design Grammars.* Journal of Information Systems, 1993. 3(4): p. 217-237.**

28. **Kamlah, W. and P. Lorenzen, *Logical Propaedeutic: Pre-School of Reasonable Discourse*. 1984: Rowman & Littlefield.**

29. **Purchase, H.C., et al., *UML Class Diagram Syntax: an Empirical Study of Comprehension,* in *Proceedings of the Australasian Symposium on Information Visualisation (InVis.au 2001)*. 2001, Australian Computer Society: Sydney, Australia. p. 113-120.**

30. **Parsons, J. and L. Cole, *An Experimental Examination of Property Precedence in Conceptual Modelling,* in *1st Asian-Pacific Conference on Conceptual Modelling (APCCM 2004)*. 2004, Australian Computer Society Inc.: Dunedin, New Zealand. p. 101-110.**

31. **Parsons, J. and Y. Wand, *Property-Based Semantic Reconciliation of Heterogeneous Information Sources,* in *21st International Conference on Conceptual Modeling (ER 2002)*. 2002, Springer: Tampere, Finland. p. 351-364.**

32. **Kim, J., J. Hahn, and H. Hahn, *How Do We Understand a System with (So) Many Diagrams? Cognitive Integration Processes in Diagrammatic Reasoning.* Information Systems Research, 2000. 11(3): p. 284-303.**

33. **Gehlert, A. and A. Heuer, *Towards Goal-Driven Self Optimisation of Service Based Applications,* in *1st International Conference of the Future of the Internet of Services (ServiceWave 2008)*. 2008, Springer: Madrid, Spain.**

34. **Gehlert, A., N. Bramsiepe, and K. Pohl, *Goal-Driven Alignment of Services and Business Requirements,* in *4th International Workshop on Service-Oriented Computing Consequences for Engineering Requirements (SOCCER 2008)*. 2008: Barcelona, Spain.**

35. **Vanderheiden, G.C., *Making Software More Accessible for People with Disabilities.* SIGCAPH Newsletter, ACM Special Interest Group on Computers and the Physically Handicapped, 1993. 47.**

36. **Stephanidis, C., et al. *Universal accessibility in HCI: Process-oriented design guidelines and tool requirements*. in *4th ERCIM Workshop on User Interfaces for All*. 1998. Stockholm.**

37. ***Disability Discrimination Act*. 1995.**

38. **SeCSE, *Specification Language Definition*. 2007.**

39. **Sawyer, P., et al., *Faceted Service Specification.* 2005.**

40. **Zachos, K., et al., *Discovering Web Services to Specify More Complete System Requirements.* Lecture notes in Computer Science, 2007. 4495: p. 142.**

41. **Alexander, C., *The timeless way of building*. 1979: Oxford University Press New York.**

42. **Sinnig, D., et al., *Practical Extensions for Task Models.* Lecture notes in Computer Science, 2007. 4849: p. 42.**

43. **Zachos, K., et al. *Seamlessly integrating service discovery into UML requirements processes*. 2006: ACM New York, NY, USA.**

44. **Fischer, G., *User Modeling in Human–Computer Interaction.* User Modeling and User-Adapted Interaction, 2001. 11(1): p. 65-86.**

45. **Cullinane, A., *Web Service Specification Development to Inform the Selection and Use in Different Organizational Cultures*. 2008, City University: London. p. 142.**

46. **SeCSE, *Quality of Service Ontology*.**

47. **Joshi, A. and S. Tripathi. *User Experience Metric and Index of Integration: Measuring Impact of HCI Activities on User Experience*.**

48. **SeCSE, *The Mechanisms for Delivering Services*. 2005.**

49. **SeCSE, *Policies Specification and Integration with Existing Standards*. 2006.**

50. **SeCSE, *Form-based language and patterns for monitoring requirements*. 2007.**