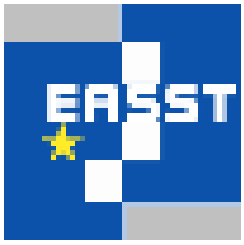
THE EASST NEWSLETTER
Volume 12
March 2006



European Association of Software Science and Technology

Special Issue
FRCSS'06: Future Research Challenges for Software and Services

Homepage of EASST**:**
http://www.easst.org
Subscription:
EASST NEWSLETTER is distributed among the members of EASST, the *European Association of Software Science and Technology*. If you are not yet a member of EASST, but you wish to receive the EASST NEWSLETTER, then you are kindly invited to become a member! Note, there are **no** membership fees. The application form can be found on the last page.
Or apply online at http://www.easst.org
Guest Editors:
Prof. Dr. Tiziana Margaria
Universität Potsdam
Prof. Dr. Matteo Banti
Prof. Dr. José-Luis Fernandéz-Villlacañas Martín
European Commisssion, DG Information Society and Media Unit D3
Editor:
Dr. Julia Padberg
Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik
Sekr. FR 6-1, Franklinstr. 28/29
D-10587 Berlin

E-mail : padberg@cs.tu-berlin.de
URL : http://tfs.cs.tu-berlin.de/~padberg/
Tel : +49-30/314-24165
FAX : +49-30/314-23516

Foreword

This first edition of the ***EASST-EU International Workshop on Future Research Challenges for Software and Services*** aims at refining and discussing the key challenges and future directions of

- Software and Services in general, and
- Fundamental Software Engineering,
- Complexity and Self-Properties,
- Services,
- Open Source Software, and
- Industrial initiatives

as key ingredients of our Strategic Objective.

The workshop is divided in two parts with distinct character:

- the morning session is devoted to presentations from our current EU-related activities in the above fields: a presentation from our expert group for Framework VII, followed by a presentation of the NESSI platform and of a number of the current ongoing projects of the Unit. All these presentations formulate their vision of the challenges and their specific and synergetic contribution.
- The afternoon sessions are devoted to the revision of the challenges via an open participation workshop, on the basis of refereed contributions.

Thirteen contributions (Six regular papers and seven short papers) have been selected out of 21 good quality submissions, covering relevant theoretical topics as well as industrial case studies.

We thank all the members of the programme committee and all the additional referees for their careful and timely evaluation of the submitted papers.

We are also grateful to the organisers of the ETAPS conference for hosting our workshop and taking care of the many organisational aspects.

We hope that you will find this program interesting and thought-provoking and that the workshop will provide you with a valuable opportunity to share ideas with other researchers and practitioners from institutions around the world.

Vienna, April 1st, 2006

The FRCSS Co-Chairs

Tiziana Margaria
Universität Potsdam
(EASST President)

Matteo Banti
José-Luis Fernández-Villacañas Martín
European Commission, DG Information Society and Media Unit D3

Program Committee

Co-Chairs

Tiziana Margaria	Universität Potsdam	(EASST President)
José-Luis Fernández-Villacañas Martín	EU, Software Technologies	
Matteo Banti	EU, Software Technologies	

Members

Michael Butler	University of Southampton	(RODIN)
Pascal Drabik	EU, Software Technologies	
Brian Fitzgerald	Limerick University	(CALIBRE)
Karl M. Goeschka	TUWien	(DediSys)
Nikolaos Georgantes	INRIA	(AMIGO)
Rishab Aiyer Ghosh	MERIT	(CALIBRE)
Harmke de Groot	Philips	(AMIGO)
Alan Hartman	IBM Haifa	(ModelWare)
Jens Knoop	TU Wien	(ETAPS General Chair)
Charles MacMillan	EU, Software Technologies	
Neil Maiden	City University London	(SeCSE)
Mira Mezini	TU Darmstadt	(AOSD)
Philippe Millot	Thales	(ModelWare)
Simin Nadjm-Tehrani	Linköping University	(DeDiSys)
Stefano de Panfilis	Engineering	(SeCSE)
Awais Rashid	Lancaster University	(AOSD)
Alexander Romanovsky	Newcastle University	(RODIN)
Jari Veijalainen	HPI	(ASG)
Mathias Weske	HPI	(ASG)

Subreviewers

Yvonne Howard
Eva Kühn
Michel Lacroix
Stephane Lo Presti
Abdolbaghi Rezazadeh
Corina Sas
Markus Schordan
Colin Snook

Projects

Program

Welcome
Tiziana Margaria, José-Luis Fernández-Villacañas Martín

The Austrian Perspective
Prof. Steinhard, Dean TU Wien

Keynote:
The future of Software and Services
Günter Böckle, Siemens

The NESSI perspective
Stefano de Panfilis, Engineering

Project presentation of challenges

AOSD - Awais Rashid
RODIN - Alexander Romanovsky
SECSE - Stefano de Panfilis
DEDISYS - Karl Goeschka
MODELWARE - Philippe Millot
ASG - Jari Veijalainen
AMIGO - Maddy Janse
INFRAWEBS - H.-J. Nern
PYPY - Alistair Burt
MADAM - Svein Hallsteinsen

Technical Session I: From Software to Services

Maintaining large software distributions: new challenges from the FOSS era

Roberto Di Cosmo *and Berke Durak **and Xavier Leroy **and Fabio Mancinelli *and Jérôme Vouillon *

*PPS, University of Paris 7, Firstname.Lastname@pps.jussieu.fr

**INRIA Rocquencourt, Firstname.Lastname@inria.fr

Abstract. *In the mainstream adoption of free and open source software (FOSS),* distribution editors *play a crucial role: they package, integrate and distribute a wide variety of software, written in a variety of languages, for a variety of purposes of unprecedented breadth.*
Ensuring the quality of a FOSS distribution is a technical and engineering challenge, owing to the size and complexity of these distributions (tens of thousands of software packages). A number of original topics for research arise from this challenge. This paper is a gentle introduction to this new research area, and strives to clearly and formally identify many of the desirable properties that must be enjoyed by these distributions to ensure an acceptable quality level.

Keywords: Open source software, dependency management, EDOS project

1 Introduction

Managing large software systems has always been a stimulating challenge for the research field in Computer Science known as Software Engineering. Many seminal advances by founding fathers of Comp. Sci. were prompted by this challenge (see the book "Software Pioneers", edited by M. Broy and E. Denert [BD02], for an overview). Concepts such as structured programming, abstract data types, modularization, object orientation, design patterns or modeling languages (unified or not) [Szy97, GHJV94], were all introduced with the clear objective of simplifying the task not only of the programmer, but of the software engineer as well.

Nevertheless, in the recent years, two related phenomena: the explosion of Internet connectivity and the mainstream adoption of free and open source software (FOSS), have deeply changed the scenarii that today's software engineers face. The traditional organized, safe world where software is developed from specifications in a fully centralized way is no longer the only game in town. We see more and more complex software systems that are assembled from loosely coupled sources developed by programming teams not belonging to any single company, cooperating only through fast Internet connections. The availability of code distributed under FOSS licences makes it possible to reuse such code without formal agreements among companies, and without any form of central authority that coordinates this burgeoning

activity.

This has led to the appearance of the so-called *distribution editors*, who try to offer some kind of reference viewpoint over the breathtaking variety of FOSS software available today: they take care of packaging, integrating and distributing tens of thousands of software packages, very few being developed in-house and almost all coming from independent developers. We believe that the role of distribution editors is deeply novel: no comparable task can be found in the traditional software development and distribution model.

This unique position of a FOSS distribution editor means that many of the standard, often unstated assumptions made for other complex software systems no longer hold: there is no common programming language, no common object model, no common component model, no central authority, neither technical nor commercial[1].

Consequently, most FOSS distribution today simply rely on the general notion of software *package*[2]: a bundle of files containing data, programs, and configuration information, with some metadata attached. Most of the metadata information deals with *dependencies*: the relationships with other packages that may be needed in order to run or install a given package, or that conflict with its presence on the system.

We now give a general description of a typical FOSS process. In figure 1 we have an imaginary project, called `foo`, handled by two developers, Alice Torvalds and Bob Dupont, who use a common CVS or Subversion repository and associated facilities such as mailing lists at a typical FOSS development site such as Sourceforge. Open source software is indeed developed as *projects*, which may group one or more developers. Projects can be characterized by a common goal and the use of a common infrastructure, such as a common version control repository, bug tracking system, or mailing lists. For instance, the Firefox browser, the Linux kernel, the KDE and Gnome desktop environments or the GNU C compiler are amongst the largest FOSS projects and have their own infrastructures. Of course, even small bits of software like `sysstat` consitute projects, even if they are developed by only one author without the use of a version control system. A given project may lead to one or more *products*. For instance, the KDE project leads to many products, from the `konqueror` browser to the desktop environment itself. Each FOSS product may then be included in a distribution. In our example, the project `foo` delivers the products `gfoo`, `kfoo` and `foo-utils`. A *port* is the inclusion of a product into a distribution by one or more *maintainers* of that distribution. The maintainers must:

- Import and regularly track the source code for the project into the distribution's own version control or storage system (this is depicted in figure 1 by a switch controlling the flow of information from the upstream to the version control system of the distribution).
- Ensure that the dependencies of the product are already included in the distribution.
- Write or include patches to adapt the program to the distribution.
- Write installation, upgrading, configuration and removal scripts.
- Write metadata and control files.

[1] In the world of Windows-based personal computing, for example, the company controlling Windows can actually impose to the ISV the usage of its API and other rules.

[2] Not to be mistaken for the software organizational unit present in many modern programming languages.

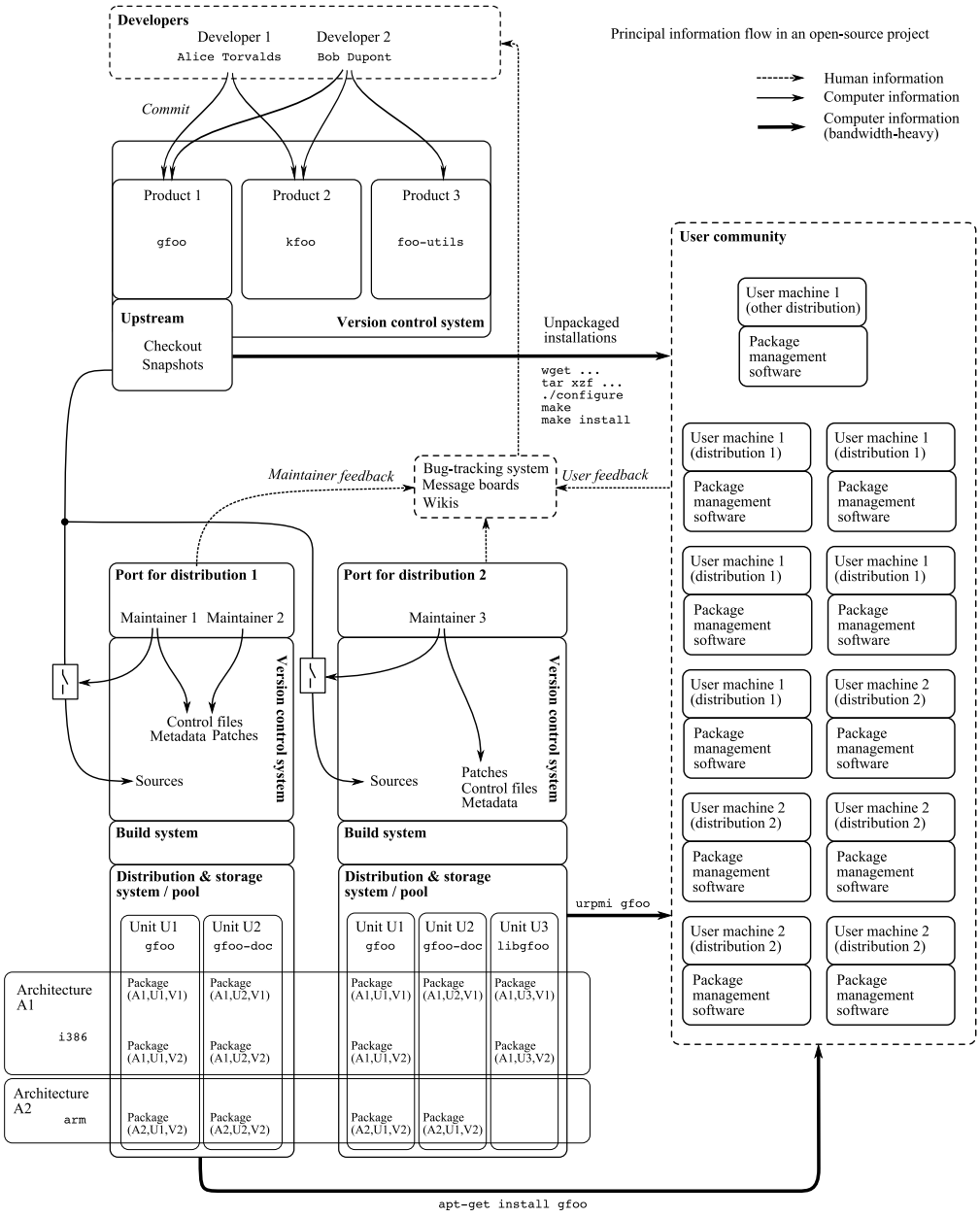


Figure 1: Major flow of information in a FOSS project.

- Communicate with the upstream developers by forwarding them bug reports, patches or feature requests.

We see that the job of maintainers is substantial for which attempts to automate some of those tasks, such as automated dependency extraction tools [Tuu03, TT01] or getting source code updates from developers [Ekl05] are no substitute. In our example, we have a Debian-based distribution 1, with two maintainers for `foo`, and an RPM-based distribution 2 with one maintainer. A given product will be divided into one or more *units*, which will be compiled for the different *architectures* supported by the distribution (a given unit may not be available on all architectures) and bundled as *packages*. The metadata and control files specify how the product is divided into units, how each unit is to be compiled and packaged and on which architectures, as well as the dependency information, the textual description of the units, their importance, and classification tags. These packages are then automatically downloaded (as well as their dependencies) by the package management software (for instance, `apt` or `urpmi`) of the users of that distribution. Some users may prefer to download directly the sources from the developers, in which case they will typically execute a sequence of commands such as `./configure && make && make install` to compile and install that software. However, they then lose the many benefits of a package management system, such as tracking of the files installed by the package, automated installation of the dependencies, local modifications and installation scripts.

We now turn to the problem of ensuring the quality of a distribution. This problem is the focus of the European FP6 project EDOS (Environment for the development and Distribution of Open Source software). This problem can therefore be divided into three main tasks:

Upstream tracking makes sure that the package in the distribution closely follows the evolution of the software development, almost always carried over by some team outside the control of the distributor.

Testing and integration makes sure that the program performs as expected in combination with other packages in the distribution. If not, bug reports need propagating to the upstream developer.

Dependency management makes sure that, in a distribution, packages can be installed and user installations can be upgraded when new versions of packages are produced, while respecting the constraints imposed by the dependency metadata.

In this paper, we focus on the last task: dependency management. This task is surprisingly complex [Tuu03, vdS04], owing to the large number of packages present in a typical distribution and to the complexity and richness of their interdependencies. It is at the very heart of the research activity conducted in workpackage 2 of the EDOS project.

More specifically, our focus is on the issues related to dependency management for large sets of software packages, with a particular attention to what must be done to maintain consistency of a software distribution *on the repository side*, as opposed to maintaining a set of packages installed *on a client machine*.

This choice is justified by the following observation: maintaining consistency of a distribution of software packages is *fundamental* to ensure quality and scalability of current and future distributions; yet, it is also an *invisible* task, since the smooth working it ensures on the end user side tends to be considered

as normal and obvious as the smooth working of packet routing on the Internet. In other words, we are tackling an essential *infrastructure* problem that has long been ignored: while there are a wealth of client-side tools to maintain a user installation (`apt`, `urpmi`, `smart` and many others [Sil04, Man05, Nie05]), there is surprisingly little literature and publically available tools that address server-side requirements. We found very little significant prior work in this area, despite it being critical to the success of FOSS in the long term.

The paper is organised as follows. Section 2 contains a formal description of the main characteristics of a software package found in the mainstream FOSS distributions, as far as dependency are concerned. In Section 3 we identify and formally define three desirable properties of a distribution with respect to dependency management. Section 4 discusses the feasibility of checking these properties. A few empirical measurements are given in section 5, followed by conclusions in section 6.

2 Basic definitions

Every package management system [DG98, Bai97] takes into account the interrelationships among packages (to different extents). We will call these relationships *requirements*. Several kinds of requirements can be considered. The most common one is a *dependency* requirement: in order to install package P_1, it is necessary that package P_2 is installed as well. Less often, we find *conflict* requirements: package P_1 cannot coexist with package P_2.

Some package management systems specialize these basic types of requirements by allowing to specify the *timeframe* during which the requirement must be satisfied. For example, it is customary to be able to express *pre-dependencies*, a kind of dependency stating that a package P_1 needs package P_2 to be present on the system *before* P_1 can be installed [DG98].

In the following, we assume the distribution and the architecture are fixed. We will identify packages, which are archive files containing metadata and installation scripts, with pairs of a unit and a version.

Definition 1 (Package, unit). *A* package *is a pair (u, v) where u is a unit and v is a version. Units are arbitrary strings, and we assume that versions are non-negative integers.*

While the ordering over version strings as used in common OSS distributions is not discrete (i.e., for any two version strings v_1 and v_2 such that $v_1 < v_2$, there exists v_3 such that $v_1 < v_3 < v_2$), taking integers as version numbers is justified for two reasons. First, any given repository will have a finite number of packages. Second, only packages with the same unit will be compared.

For instance, if our Debian repository contains the versions `2.15-6`, `2.16.1cvs20051117-1` and `2.16.1cvs20051206-1` of the unit `binutils`, we may encode these versions respectively as 0,1 and 2, giving the packages $(\texttt{binutils}, 0)$, $(\texttt{binutils}, 1)$, and $(\texttt{binutils}, 2)$.

Definition 2 (Repository). *A* repository *is a tuple $R = (P, D, C)$ where P is a set of packages, $D : P \to \mathscr{P}(\mathscr{P}(P))$ is the dependency function*[3]*, and $C \subseteq P \times P$ is the conflict relation. The repository must satisfy the following conditions:*

- *The relation C is symmetric, i.e., $(\pi_1, \pi_2) \in C$ if and only if $(\pi_2, \pi_1) \in C$ for all $\pi_1, \pi_2 \in P$.*

[3] We write $\mathscr{P}(X)$ for the set of subsets of X.

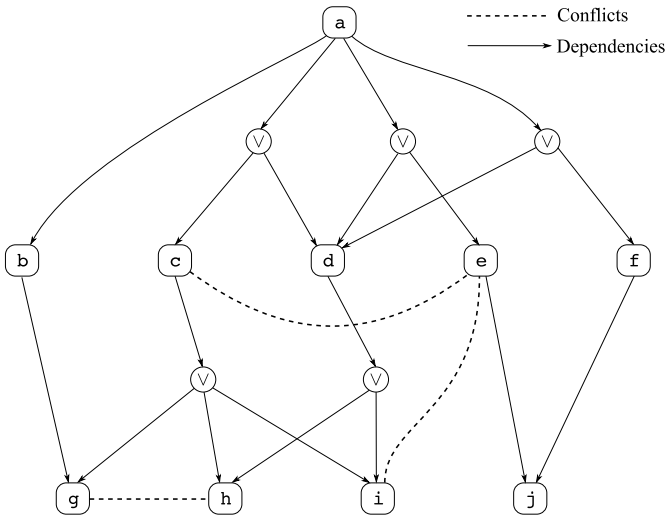


Figure 2: The repository of example 1.

- *Two packages with the same unit but different versions conflict*[4]*, that is, if* $\pi_1 = (u, v_1)$ *and* $\pi_2 = (u, v_2)$ *with* $v_1 \neq v_2$*, then* $(\pi_1, \pi_2) \in C$*.*

In a repository $R = (P, D, C)$, the dependencies of each package p are given by $D(p) = \{d_1, \ldots, d_k\}$ which is a set of sets of packages, interpreted as follows. If p is to be installed, then all its k dependencies must be satisfied. For d_i to be satisfied, at least one of the packages of d_i must be available. In particular, if one of the d_i is the empty set, it will never be satisfied, and the package p is not installable.

Example 1. Let $R = (P, D, C)$ be the repository given by

$$
\begin{aligned}
&P = \{a, b, c, d, e, f, g, h, i, j\} \\
&D(a) = \big\{\{b\}, \{c, d\}, \{d, e\}, \{d, f\}\big\} \\
&D(b) = \big\{\{g\}\big\} \quad D(c) = \big\{\{g, h, i\}\big\} \quad D(d) = \big\{\{h, i\}\big\} \\
&D(e) = D(f) = \big\{\{j\}\big\} \\
&C = \{(c, e), (e, c), (e, i), (i, e), (g, h), (h, g)\}
\end{aligned}
$$

where $a = (u_a, 0)$, $b = (u_b, 0)$, $c = (u_c, 0)$ and so on. The repository R is represented in figure 2. For the package a to be installed, the following packages must be installed: b, either c or d, either d or e, and either d or f. Packages c and e, e and i, and g and h cannot be installed at the same time.

In computer science, dependencies are usually *conjunctive*, that is they are of the form

$$a \to b_1 \wedge b_2 \wedge \cdots \wedge b_s$$

[4]This requirement is present in some package management systems, notably Debian's, but not all. For instance, RPM-based distributions allow simultaneous installation of several versions of the same unit, at least in principle.

where a is the target and b_1, b_2, ... are its prerequisites. This is the case in `make` files, where all the dependencies of a target must be built before building the target. Such dependency information can be represented by a directed graph, and dependencies can be solved by the well-known topological sort algorithm. Our dependencies are of a more complex kind, which we name *disjunctive* dependencies. Their general form is a conjunction of disjunctions:

$$a \to (b_1^1 \vee \cdots \vee b_1^{r_1}) \wedge \cdots \wedge (b_s^1 \vee \cdots \vee b_s^{r_s}). \tag{1}$$

For a to be installed, each term of the right-hand side of the implication 1 must be satisfied. In turn, the term $b_i^1 \vee \cdots \vee b_i^{r_i}$ when $1 \leq i \leq s$ is satisfied when at least one of the b_i^j with $1 \leq j \leq r_i$ is satisfied. If a is a package in our repository, we therefore have

$$D(a) = \{\{b_1^1, \ldots, b_1^{r_1}\}, \cdots, \{b_s^1, \ldots, b_s^{r_s}\}\}.$$

In particular, if one of the terms is empty (if $\varnothing \in D(a)$), then a cannot be satisfied. This side-effect is useful for modeling repositories containing packages mentioning another package b that is not in that repository. Such a situation may occur because of an error in the metadata, because the package b has been removed, or b is in another repository, maybe for licensing reasons.

Concerning the relation C, two packages $\pi_1 = (u_1, v_1), \pi_2 = (u_2, v_2) \in P$ conflict when $(\pi_1, \pi_2) \in C$. Since conflicts are a function of presence and not of installation order, the relation C is symmetric.

Definition 3 (Installation). *An* installation *of a repository $R = (P, D, C)$ is a subset of P, giving the set of packages installed on a system. An installation is* healthy *when the following conditions hold:*

- ***Abundance:*** *Every package has what it needs. Formally, for every $\pi \in I$, and for every dependency $d \in D(\pi)$ we have $I \cap d \neq \varnothing$.*
- ***Peace:*** *No two packages conflict. Formally, $(I \times I) \cap C = \varnothing$.*

Definition 4 (Installability and co-installability). *A package π of a repository R is* installable *if there exists a healthy installation I such that $\pi \in I$. Similarly, a set of packages Π of R is* co-installable *if there exists a healthy installation I such that $\Pi \subseteq I$.*

Note that because of conflicts, every member of a set $X \subseteq P$ may be installable without the set X being co-installable.

Example 2. Assume a depends on b, c depends on d, and c and d conflict. Then, the set $\{a, b\}$ is not co-installable, despite each of a and b being installable and not conflicting directly.

Definition 5 (Maximal co-installability). *A set X of co-installable packages of a repository R is* maximal *if there is no other co-installable subset X' of R that strictly contains X. We write $\mathrm{maxco}(R)$ for the family of all maximal co-installable subsets of R.*

Definition 6 (Dependency closure). *The* dependency closure $\Delta(\Pi)$ *of a set of package Π of a repository R is the smallest set of packages included in R that contains Π and is closed under the* immediate dependency *function $\overline{D} : \mathscr{P}(P) \to \mathscr{P}(P)$ defined as*

$$\overline{D}(\Pi) = \bigcup_{\substack{\pi \in \Pi \\ d \in D(\pi)}} d.$$

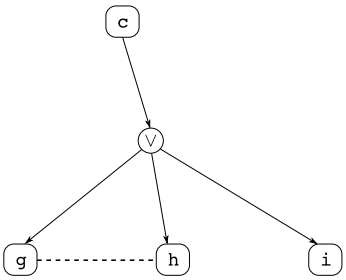


Figure 3: The subrepository generated by package c. The dependency closure is $\{c, g, h, i\}$.

In simpler words, $\Delta(\Pi)$ contains Π, then all packages that appear as immediate dependencies of Π, then all packages that appear as immediate dependencies of immediate dependencies of Π, and so on. Since the domain of $\overline{D}$ is a complete lattice, and $\overline{D}$ is clearly a continuous function, we immediately get (by Tarski's theorem) that such a smallest set exists and can be actually computed as follows:

Proposition 1. *The dependency closure* $\Delta(\Pi)$ *of* Π *is:*

$$\Delta(\Pi) = \bigcup_{n \geq 0} \overline{D}^n(\Pi).$$

The notion of dependency closure is useful to extract the part of a repository that pertains to a package or to a set of packages.

Definition 7 (Generated subrepository). *Let* $R = (P, D, C)$ *be a repository and* $\Pi \subseteq P$ *be a set of packages. The* subrepository generated by Π *is the repository* $R|_\Pi = (P', D', C')$ *whose set of packages is the dependency closure of* Π *and whose dependency and conflict relations are those of* R *restricted to that set of packages. More formally we have* $P' = \Delta(\Pi)$, $D' : P' \to \mathscr{P}(\mathscr{P}(P')), \pi \mapsto \{d \cap P' \mid d \in D(\pi)\}$ *and* $C' = C \cap (P' \times P')$.

Figure 3 shows the subrepository generated by the package c of example 1. The dependency closure of c is the set of package nodes of that subrepository.

We then have the following property, which allows to consider only the relevant subrepositories when answering questions of installability.

Proposition 2 (Completeness of subrepositories). *A package* π *is installable w.r.t.* R *if and only if it is installable w.r.t.* $R|_\pi$. *(Similarly for co-installability.)*

3 Maintaining a package repository

The task of maintaining a package repository is difficult: the maintainance team must monitor the evolution of thousand of packages over time, and address the error reports coming from different sources

(users, QA teams, developers, etc.). It is desirable to automate as much of this work as possible. Our medium-term goal is to build tools that help distribution maintainers track dependency-related problems in package repositories. We detail here some of the desirable properties of a repository. The first is *history-free*, in that it applies to a given state of a repository.

Being trimmed We say that a repository R is *trimmed* when every package of R is installable w.r.t. R. The intuition behind this terminology is that a non-trimmed repository contains packages that cannot be installed in any configuration. We call those packages *broken*. They behave as if they were not part of the repository. It is obviously desirable that at any point in time, a repository is trimmed, that is, contains no broken packages.

The next properties are *history-sensitive*, meaning that they take into account the evolution of the repository over time. Due to this dependency on time, the precise formulation of these properties is delicate. Just like history-free properties are relevant to users who install a distribution from scratch, history-sensitive properties are relevant to users who upgrade an existing installation.

Monotonicity Let R_t be the repository at time t and consider a coinstallable set of packages C_t. Some users can actually have packages C_t installed simultaneously on their system. These users have the possibility of installing additional packages from R_t, resulting in a coinstallable set of packages C'_t. These users can reasonably expect that they will be able to do so (extend C_t into C'_t) at any future time t', using the repository $R_{t'}$, which, being *newer*, is supposed to be *better* than the old R_t.

Of course, users are ready to accept that in $R_{t'}$ they will not get exactly ${C'}_t$, but possibly ${C'}_{t'}$, where some packages were updated to a greater version, and some others have been replaced as the result of splitting into smaller packages or grouping into larger ones. But, clearly, it is not acceptable to evolve R_t into $R_{t'}$ if R_t allows to install, say, `apache` together with `squid`, while $R_{t'}$ does not.

We say that a repository history line is *monotone* if the *freedom* of a user to install packages is a monotone function of time. Writing $F(x, R)$ for the set of possible package sets in R that are a possible replacement of package x according to the metadata, monotonicity can be formally expressed as

$$\mathrm{Mon}(R) = \forall t < t'.\, \forall P \in \mathrm{Con}(R_t).\, \exists Q \in \mathrm{Con}(R_{t'}).\, \forall x \in P.\, Q \cap F(x, R_{t'}) \neq \varnothing$$

Upgradeability Another reasonable expectation of the user is to be able to upgrade a previously installed package to the most recent version (or even any more recent version) of this package that was added to the repository since her latest installation. She is ready to accept that this upgrade will force the installation of some new packages, the upgrade of some other packages, and the replacement of some sets of package by other sets of packages, as the result of the reorganization of the structure of the packages.

She may even accept, in order to perform an important upgrade, to see some previously installed packages removed, as it happens when using all the meta package management tools available today.

We remark that these properties are *not* interdefinable. We give here a proof of this assertion by exhibiting example repositories showing this independence of the properties. For the first two cases,

consider three repositories R_1, R_2, R_3 whose sets of packages are $P_1 = \{(a,1),(b,1),(c,1)\}$, $P_2 = \{(a,1),(b,1)\}$, $P_3 = \{(a,1),(a,2),(b,1)\}$ with no conflicts nor dependencies among the version 1 packages and a conflict among $(a,2)$ and $(b,1)$. Notice that at each moment t in time, R_t is trimmed.

1. A repository that stays trimmed over a period of time is not necessarily monotone, nor upgradeable. Since $(c,1)$ disappears between times 1 and 2, this step in the evolution does not preserve monotonicity. Since $(a,2)$ has a new conflict (namely with $(b,1)$) in R_3, the evolution from R_2 to R_3 does not preserve upgradeability.

2. A repository that stays trimmed over a period of time and evolves in a monotone fashion is not necessarily upgradeable. The evolution from R_2 to R_3 above is monotone, each of R_2 and R_3 is trimmed, but we fail upgradeability because there is no way of going from $\{(a,1),(b,1)\}$ to $\{(a,2),(b,1)\}$ because of the conflict.

3. A repository that stays trimmed over a period of time and is upgradeable is not necessarily monotone.

   Consider repositories R_1 and R_2 with $P_1 = \{(a,1),(b,1)\}$ and $P_2 = \{(a,2),(b,1)\}$. Assume $(a,1)$ and $(b,1)$ are isolated packages, while $(a,2)$ conflicts with $(b,1)$. Now, a user having installed all of R_1 and really willing to get $(a,2)$ can do it, but at the price of giving up $(b,1)$. This evolution of the repository is therefore upgradeable but not monotone.

4. A repository that evolves in a monotone and upgradeable fashion is not necessarily trimmed at any time: indeed, the monotonicity and upgradeability property only speak of *consistent* subsets of a repository, that cannot contain, by definition, any broken packages.

   Consider for example repositories R_1, R_2 with $P_1 = \{(a,1)\}$, $P_2 = \{(a,1),(b,1)\}$. Assume $(a,1)$ and $(b,1)$ are broken because they depend on a missing package $(c,1)$. Here, the evolution of R_1 to R_2 is trivially monotone and upgradeable, because there is *no* consistent subset of R_1 and R_2, and both R_1 and R_2 are not trimmed because they contain broken packages.

The examples above to prove that the three properties are actually independent may seem contrived, but are simplifications of real-world scenarii. For instance, example 3 can actually happen in the evolution of real repositories, when for some reason the new version of a set of interrelated packages is only partially migrated to the repository. Many packages are split into several packages to isolate architecture-independent files, as in the Debian packages `swi-prolog` and `swi-prolog-doc`. When performing this split, it is quite natural to add a conflict in `swi-prolog-doc` against old, non-splitted versions of `swi-prolog`. If the new version of `swi-prolog-doc` slips into a real repository before the new, splitted version of `swi-prolog`, we are exactly in situation number 3 above.

Package developers seem aware of some of these issues: they actually do their best to ensure monotonicity and upgradeability by trying to reduce as much as possible the usage of conflicts, and sometime resorting to naming conventions for the packages when a radical change in the package happens, like in the case of `xserver-common` vs. `xserver-common-v3` in Debian, as can be seen in the dependencies for `xserver-common`.

```
Package: xserver-common
Conflicts: xbase (<< 3.3.2.3a-2), xsun-utils, xbase-clients (<< 3.3.6-1),
  suidmanager (<< 0.50), configlet (<= 0.9.22),
  xserver-3dlabs (<< 3.3.6-35), xserver-8514 (<< 3.3.6-35),
  xserver-agx (<< 3.3.6-35), xserver-common-v3 (<< 3.3.6-35),
  xserver-fbdev (<< 3.3.6-35), xserver-i128 (<< 3.3.6-35),
  xserver-mach32 (<< 3.3.6-35), xserver-mach64 (<< 3.3.6-35),
  xserver-mach8 (<< 3.3.6-35), xserver-mono (<< 3.3.6-35),
  xserver-p9000 (<< 3.3.6-35), xserver-s3(<< 3.3.6-35),
  xserver-s3v (<< 3.3.6-35), xserver-svga (<< 3.3.6-35),
  xserver-tga (<< 3.3.6-35), xserver-vga16 (<< 3.3.6-35),
  xserver-w32 (<< 3.3.6-35), xserver-xsun (<< 3.3.6-35),
  xserver-xsun-mono (<< 3.3.6-35), xserver-xsun24 (<< 3.3.6-35),
  xserver-rage128, xserver-sis
```

4 Algorithmic considerations

Our research objective within the EDOS project is to formally define the desirable properties of repositories stated in section 3 (and possibly other properties that will appear useful), and to develop efficient algorithms to check these properties automatically.

It is really not evident that any of these problems are actually tractable in practice: due to the rich language allowed to describe package dependencies in the mainstream FOSS distributions, even the simplest problems (checking installability of a single package) may involve verifications over a large number of other packages. During our first investigations of these problems, we have indeed already proven the following complexity result.

Theorem 1 (Package installability is an NP-complete problem). *Checking whether a single package P can be installed, given a repository R, is NP-complete.*

The full proof of this result will be published separately. It relies on a simple, polynomial-time reduction of the 3SAT problem to the installability problem. Given an instance of 3SAT, a repository is constructed having one package for the whole 3SAT formula, one package per clause of that formula, and three packages for each propositional atom occurring in that formula. Dependencies and conflicts between these packages are added in such a way that the package for the whole formula is installable if and only if the 3SAT formula is satisfiable.

Nevertheless, this strong limiting result does not mean that we will not be able to decide installability and the other problems in practice: the actual instances of these problems, as found in real repositories, could be quite simple in the average.

In particular, the converse of the reduction used for the NP-completeness proof leads to an effective way of deciding package installability. We developed an algorithm that encodes a repository R and its dependencies as a Boolean formula $C(R)$. (Details of the encoding will be published in a forthcoming paper.) Assignments of truth values to boolean variables that satisfy $C(R)$ are in one-to-one correspondence with sets of co-installable packages. Therefore, a package P is installable if and only if the Boolean formula $C(R) \wedge P$ is satisfiable, which we can check relatively efficiently using off-the-shelf SAT solving

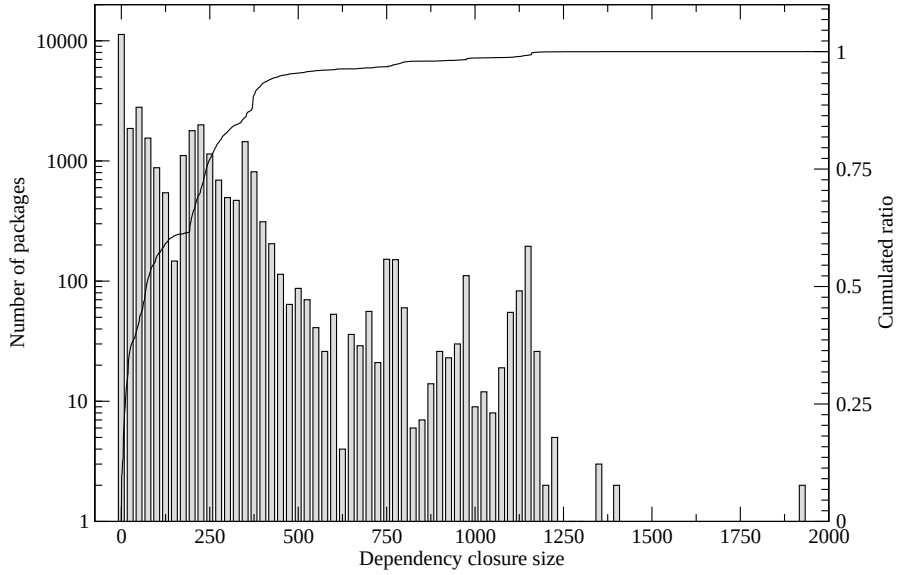


Figure 4: Number of packages as a function of the size of their dependency closures.

technology.

We implemented the conversion algorithm as well a SAT solver [ES04] and ran it over both the Debian pool (over 30,000 packages) and the Mandriva Cooker distribution (around 5,000 packages). The execution time is entirely acceptable, and the tool found a number of non-installable packages in both distributions.

We are now focusing our attention on the two time-dependent desirable properties for the repositories, which are, algorithmically speaking, much harder.

5 Empirical measurements

In parallel with our formal complexity and algorithmic investigations, we also performed some empirical measurements on the Debian and Mandriva distributions, to try and grasp the practical complexity of the problems.

Figure 4 gives a histogram showing the number of packages as a function of the size of the dependency closure, from the Debian stable, unstable and testing pools on 2005-12-13, which has 31149 packages. The average closure size is 158; 50% of the packages have a closure size of 71 or less, 90% of 372 or less, and 99% of 1077 or less. These numbers show that naive combinatorial algorithms, exponential in the size of the dependency closure, are clearly out of the question.

Figure 5 estimates the complexity of solving the Boolean formulae generated by our encoding of the installability problem. The "temperature" T of a formula in 3SAT conjunctive normal form is defined as $T = m/n$ where m is the number of clauses and n the number of variables. There is strong theoretical

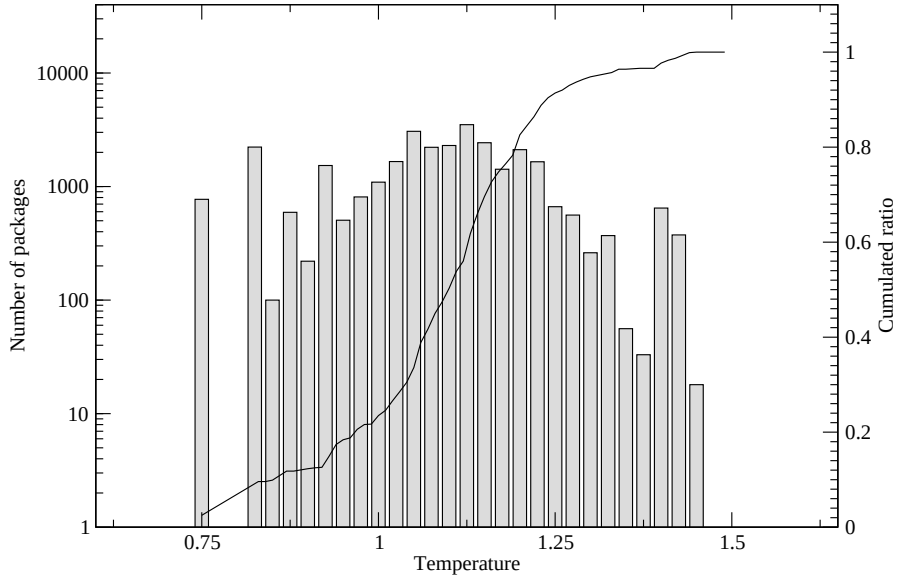


Figure 5: Number of packages as a function of the “temperature” of the SAT problems corresponding to their installability problems.

and practical evidence that hard SAT problems have a temperature close to 4.2, while SAT problems with temperatures well below or above that limit are easier to solve. The temperatures for the SAT problems corresponding to installability of the Debian packages range from 0.75 to 1.49, well below the threshold value of 4.2. This result confirms that we are dealing with relatively easy satisfiability problems, maybe owing to the small-world nature of the dependency graphs [LW05].

6 Conclusions

We have presented and motivated in this paper three fundamental properties for large repositories of FOSS packages that are quite different from the usual properties of component collections, due to the large spectrum of languages, technologies, frameworks and interfaces spanned by a contemporary FOSS distribution.

Despite their algorithmic complexity, we have already performed large-scale tests indicating that the first of these properties can be mechanically checked in reasonable time. We continue similar investigations on the other properties.

We claim that providing efficient tools to check these properties is an essential step in order to ensure that the FOSS development model stays sustainable, and we suggest that researchers should look into the specificities brought by FOSS in the software engineering world.

References

[Bai97] Edward C. Bailey. Maximum RPM, taking the Red Hat package manager to the limit. http://rikers.org/rpmbook/,http://www.rpm.org, 1997.

[BD02] Manfred Broy and Ernst Denert. *Software Pioneers: Contributions to Software Engineering*. Springer-Verlag, 2002.

[DG98] Debian Group. Debian policy manual. http://www.debian.org/doc/debian-policy/, 1996–1998.

[Ekl05] David Eklund. The lib update/autoupdate suite. http://luau.sourceforge.net/, 2003–2005.

[ES04] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[LW05] Nathan LaBelle and Eugene Wallingford. Inter-package dependency networks in open-source software. *Submitted to Journal of Theoretical Computer Science*, 2005.

[Man05] Mandriva. URPMI. http://www.urpmi.org/, 2005.

[Nie05] Gustavo Niemeyer. Smart package manager. http://labix.org/smart/, 2005.

[Sil04] Gustavo Noronha Silva. Apt-howto. http://www.debian.org/doc/manuals/apt-howto/, 2004.

[Szy97] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison Wesley Professional, 1997.

[TT01] L. Taylor and L. Tuura. Ignominy: a tool for software dependency and metric analysis with examples from large HEP packages. In *Proceedings of CHEP'01*, 2001.

[Tuu03] L. A. Tuura. Ignominy: tool for analysing software dependencies and for reducing complexity in large software systems. In *Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, volume 502, pages 684–686, 2003.

[vdS04] Tijs van der Storm. Variability and component composition. In *Proceedings of the Eighth International Conference on Software Reuse (ICSR-8)*, 2004.

Softure: Adaptable, Reliable and Performing Software for the Future

Antonia Bertolino*, Wolfgang Emmerich,**
Paola Inverardi*, Valérie Issarny*****
*CNR, Italy, **UCL, UK,
***University of L'Aquila, Italy, ****INRIA, France

Abstract. *This paper discusses the approach that will be taken by the PLASTIC project (http://www.ist-plastic.org) in order to assist the development of adaptable, reliable and performing software services for Beyond 3rd Generation networks.*

Keywords: B3G network, service-oriented architecture, middleware, validation.

1 Overview

Software in the future (Softure) will need to cope with variability, as software systems get deployed on an increasingly large diversity of computing platforms. Software will have to be usable in various environments, due to tremendous evolution in information and communication technologies. Heterogeneity of the underlying communication and computing infrastructure, mobility and continuously evolving requirements demand new software paradigms that span the entire life-cycle from development to deployment and execution. Softure must be developed in a way that facilitates both its deployment over heterogeneous networks of heterogeneous nodes, and its interaction with end users, their environment and/or other existing systems, depending on the application domain. Moreover, Softure should be reliable and meet the user's performance requirements and needs. Softure can greatly differ in nature, varying from complex and distributed software systems for highly dynamic networks of mobile nodes to embedded software systems for wireless, resource-constrained nodes. Additionally, the user-centric dimension of the new emerging applications requires Softure to be adaptive to a context that combines user-centric data (e.g., what is the information of interest for the user given his/her current situation?) and resource/computer-centric data (e.g., what is the service that can be delivered to the user given available energy?). Finally, due to its pervasiveness, Softure must be dependable, which is made more complex given the highly dynamic nature of service provision.

Supporting the development and execution of Softure systems raises numerous challenges, from elaborating languages, methods and tools for the systems' thorough design and validation in order to ensure dependability of the self-adaptive systems that are targeted, to developing supporting middleware infrastructures in order to ease the implementation and deployment of the target systems on highly heterogeneous and dynamic platforms. The next section discusses in more details the challenges that arise to support the development of dependable, adaptable Softure systems. Section 3

then describes the approach undertaken in the IST PLASTIC project for a specific instance of Softure focused on software for Beyond 3rd Generation (B3G) networks. Section 4 presents concluding remarks.

2 Research Challenges for Softure

Various abstractions for modeling behavioral adaptation of applications in response to changes in the executing environment have been proposed recently, ranging from resource aware programming to adaptive software architectures.

2.1 Developing Adaptable Applications

Proposed solutions to supporting the development of adaptable applications include approaches for the customization of the source code. For example, aspect oriented programming (AOP) for conventional computing infrastructures has gained popularity to increase programming flexibility [1]. More advanced solutions propose context-aware programming for ad hoc mobile environments [2]. Overall, the objective of these approaches is to assist the development of applications that are *generic* and can be adapted with respect to a dynamically provided context, which is in particular characterized in terms of available (hardware or software) resources, each one with its own characteristics. It is then crucial to enforce correctness of the adaptive, generic applications, which can be achieved using a declarative and deductive approach that enables the construction of generic adaptable application code and its correct adaptation with respect to a given execution context [6, 7]. Yet another approach to adaptability of software applications is presented in [3, 4], which focus on correct dynamic linking rather than on correct tailoring of generic applications. Specifically, these papers address dynamic software update using verifiable native code, such as Proof Carrying Code (PCC), to deliver correct patches that may be applied at runtime to software, according to availability requirements. The approach requires the code to be written so that it can be dynamically updated. It deals with dynamic linking in order to patch on the fly executable code, and with code verification in order to ensure that the received patch is correct with respect to some safety properties. Along this line of research, it is worthwhile mentioning the research project on Resource Aware Programming [5] ended in 2005, which focuses on functional programming and on programming in presence of bounded resources in the more confined context of embedded software. The newly started Global Computing IP project MOBIUS [38] is also of relevance although focusing on security. Still, the security attribute is taken in its full generality and accounts also for resource usage and management and for distribution and mobility. MOBIUS proposes the use of PCC techniques integrated with type theory in order to attach to (mobile) components certificates that state the security and safety properties of the dynamic behavior of the component.

Adaptation of a software system may also be addressed in a compositional way. In that direction, software architectures are very effective tools for the description and the modeling of complex systems in terms of the composition of component systems [8]. Software architectures support the description of the static and dynamic components of the system together with how they interact. Software architectures are the basis for early analysis, verification and validation of software systems. Software

architectures are the earliest comprehensive system model in the software lifecycle built from requirements specification. They are increasingly part of standardized software development processes because they represent a system abstraction in which design choices relevant to the correctness of the final system are taken. Software architectures can be used for validating the final system with respect to architectural properties [39]. Moreover, software architecture modeling is also particularly useful for assessing, so-called non-functional requirements, notably dependability requirements such as security and reliability [40] and quantitative requirements like performance and scalability [41]. Therefore, much of software architecture research has been concentrated on the specification and analysis of both qualitative and quantitative system properties. The state of the art in the field of software architectures for dependable self-adaptive distributed systems is still preliminary and fragmented, focusing on a small set of the systems' attributes. Software architectures for self-adaptive systems can be found in the literature, e.g., [9-12]. However, proposed approaches still lack associated design and validation methodologies and further address very specific adaptation with respect to computer-centric context awareness. Comprehensive design and validation of self-adaptive systems require accounting for both functional and non-functional properties in front of highly dynamic environments. Some proposals in this direction are emerging for service oriented architectures [42]. In addition, adaptation applies to both the application and middleware layers, regarding both the overall distributed systems and component embedded systems.

2.2 Middleware Supporting Adaptation

As argued above, development of adaptive software systems shall be accounted for at the middleware layer, which must both adapt its behavior according to context (i.e., available computing and networking resources and application requirements) and provide relevant feedback about the underlying infrastructure to the application layer. The former requirement may be addressed using component-based middleware, which allows enforcement of the required quality of service through the integration of adequate middleware-related services [13]. However, customization of the middleware is mostly addressed at design time, including possible middleware adaptation to deal with environmental changes [14]. To effectively support the development of Softure, context-aware composition must be enabled anytime, anywhere and hence must not rely on a priori knowledge of the computing environment, and in particular of available service instances. In other words, while most existing middleware systems require application developers to specify the instances of (middleware- and application-related) services to be used in the composition of applications, this composition shall be automated with respect to the context in Softure. Further, middleware for Softure will deal with the heterogeneity of the networking and computing environment. This includes both (i) addressing the integration of available middleware that offer effective support for their target application domains, and (ii) devising novel, advanced middleware for emerging application domains and/or infrastructures. Solutions to the former issue range from providing a new middleware API that allows benefiting from the various functionalities of the integrated middleware (e.g., [15, 16]) to implementing transparent mapping from one middleware to another at the network layer (e.g., [17]). The latter issue ranges from devising new solutions to meeting traditional non-functional requirements in order to cope with the evolving computing environment (e.g., trust management to deal with privacy and security requirements in the open networking environment [18]), to introducing new middleware architectures

to cope with the specifics of emerging networking infrastructure and devices (e.g., B3G networks [19], sensor networks [20]).

Many middleware systems have concentrated on abstracting the complexities of supporting context-awareness [21-23], by providing transparent communication mechanisms in a pervasive environment, and resource management that supports adaptations to the particular domain in which the infrastructure operates. The approaches on which the systems are based vary from system to system, though some key middleware paradigms are favored. For instance, for decoupling software components, supporting complex communication patterns, and allowing transparent communication between objects, the publish-subscribe (distributed event service) paradigm has been used considerably [24-25]. In general, various middleware infrastructures have been proposed since the end of the last century to support the development of context-aware systems. The i-Land project supports the collaboration of people within an environment full of ubiquitous components [26]. An infrastructure called Beach [27] supports the dynamic configuration of the components. The COAST framework [25] provides Beach with the functionality to distribute, replicate and synchronize objects. Aura [28] has been developed for wearable, handheld devices, providing for nomadic, possibly intermittent file access, resource monitoring and application-aware adaptation. Gaia [29] enables the coordination of software entities and heterogeneous networked devices by extending the typical operating system concepts to include context and other basic services. Dynamic and automated composition of middleware-related services for enforcing dependability according to the context is addressed by the CANS infrastructure [30], which allows components to be injected into the network for dynamically adapting the system to resource characteristics of end devices and network links. A similar approach is undertaken by the WSAMI environment [31], which builds on the Web services architecture and realizes on-line distributed connector customization for increased availability of services.

As outlined above, providing solutions to the development of Softure that are adaptive to the rich context raises a number of challenging issues relevant to all phases of software development. And, although useful approaches have emerged since the early 2000s, open problems remain to be addressed and solutions need be integrated into a comprehensive development platform, possibly aimed at a specific application domain and/or computing infrastructure. In that direction, the following presents an overview of the research that we will undertake as part of the IST project PLASTIC (Providing Lightweight and Adaptable Service Technology for pervasive Information and Communication) project (http://www.ist-plastic.org), which has commenced in February 2006. Specifically, the PLASTIC project focuses on assisting the development of adaptable software services for B3G distributed computing platforms, in particular enforcing dependability of services. The PLASTIC consortium brings together expertise in the target technical fields: the University partners (University of L'Aquila, UCL, and University of Lugano) have unique expertise in devising novel software engineering solutions for advanced software applications, and in particular applications for next generation, wireless networks; the national research institutes partners (INRIA and CNR-Pisatel) have a long track record in investigating solutions for supporting the development of innovative applications for next generation networks; and the industrial partners (IBM, Siemens Business Services, Telefonica, Virtual Trip, Pragmatica Technologies and 4D Soft) together provide

major capabilities for the development and deployment of software services for next generation networks.

3 The PLASTIC Approach

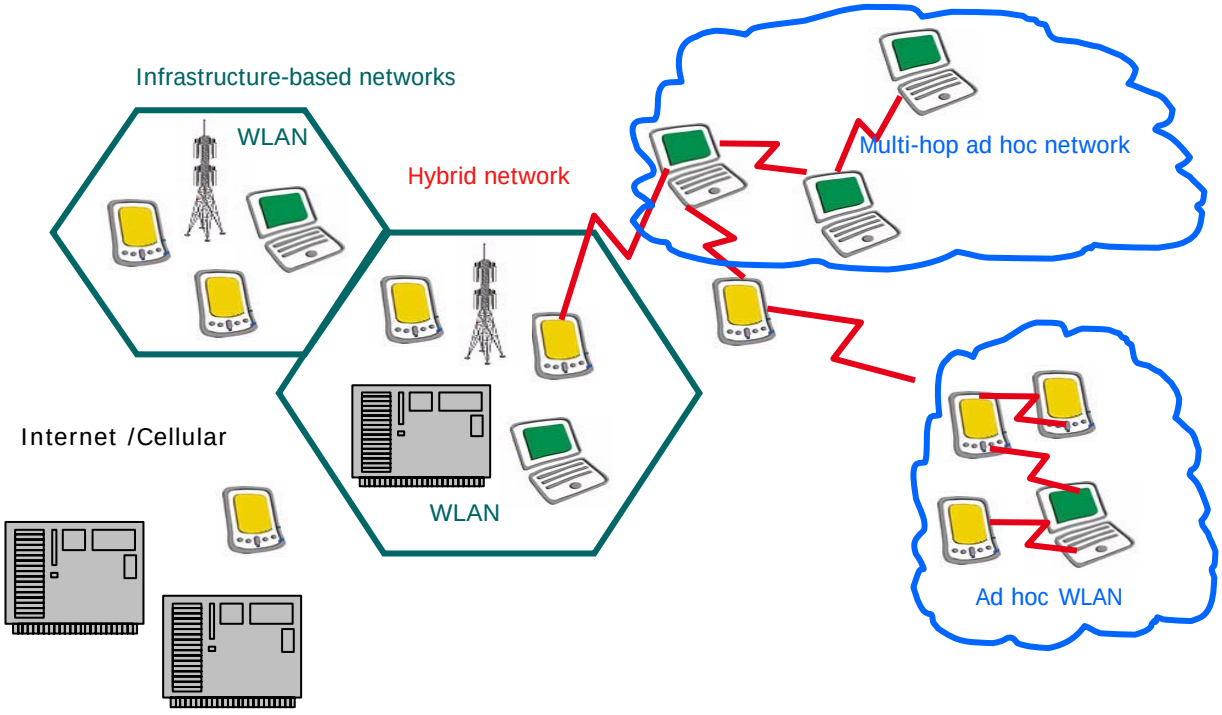


Figure 1: The B3G open wireless environment

The PLASTIC project aims to offer a comprehensive provisioning platform for software services deployed over B3G networks (see Figure 1), integrating a supporting development environment and middleware. The platform will enable dynamic adaptation of services to the environment with respect to resource availability and delivered QoS, via a development paradigm based on Service Level Agreements and resource-aware programming. The middleware will be service oriented, to enable integration and composition of heterogeneous software services from both infrastructure-based and ad hoc networks. The middleware will integrate key functions for supporting the management of adaptive services in the open wireless environment, dealing with resource awareness and dependability. The following section further discusses the PLASTIC vision, leading to introduce the supporting PLASTIC platform in Section 3.2. Sections 3.3 and 3.4 then concentrate on the two key elements of the PLASTIC platform, i.e., (i) the PLASTIC development environment enabling the comprehensive development of dependable, adaptive services, which is complemented with (ii) the PLASTIC

middleware supporting the deployment and execution of adaptive software services over the heterogeneous platforms that are networked in B3G.

3.1 The PLASTIC Vision

The vision of PLASTIC is that users in the B3G era should be provided with a variety of application services exploiting the network's diversity and richness, without requiring systematic availability of an integrated network infrastructure. The success of the provided services then depends on the user perception of the delivered Quality of Service (QoS), which varies along several dimensions, including: type of service, type of user, type of access device, and type of execution network environment. In order to manage these various factors, the network's diversity and richness must be made available and be exploitable at the application layer, where the delivered services can be most suitably adapted. This demands a comprehensive software engineering approach to the provisioning of services, which encompasses the full service life cycle, from development to validation, and from deployment to execution.

The PLASTIC answer to the above needs is to offer a comprehensive platform for the creation and provisioning of lightweight, adaptable services for the open wireless environment. Various service delivery platforms have been proposed for the 2G+ to 3G cellular networks (e.g., JAIN [32], 3GPP [33] initiatives including CAMEL, OSA and IMS, PARLAY [34]). However, these platforms are focused on network-layer services. For B3G networks, there are proposals for extending the above solutions, dealing with the provision of network-layer services that can be adapted at the middleware layer [35]. At the application layer, modeling, development and deployment tools for programming, uploading and instantiating applications and code on mobile, wireless devices have been in use worldwide by many manufacturers. The major standards in this space include J2ME [36], OSGi [37] and others. Java virtual machines, lightweight messaging systems, lightweight Web Services tool kits, small-footprint databases for device controllers and programming environments have been developed (e.g., see the software available at https://secure.alphaworks.ibm.com, which will be exploited in the development of PLASTIC software tools and middleware). In addition, much research has been conducted in the areas of mobile computing frameworks, mobile grids and environments for enabling ad hoc communication and integration. The techniques, methods, tools and programming models are still evolving. However, they are primarily focused horizontally, which is to say, on a single layer of the system's infrastructure. For example, the state of the art in adaptiveness to resources and QoS, which is a key feature of mobile adaptive services, addresses individually the network, middleware and application layers.

A key challenging contribution of the PLASTIC project is to coherently manage adaptation in a vertical way across the different layers, from application to middleware to network. This will be achieved by modeling and supporting the relevant characteristics of the various heterogeneous infrastructures, so that they are made visible and manageable to the application layer through an integrated service development and execution platform. Key research points are:

- The identification of the fair tradeoffs among resources to be made visible at the application and middleware layers, and adaptation capabilities of the service;

- The ability to maintain QoS through adaptation.

The core objective of the PLASTIC project is to enable the development and deployment of cost-effective application services, both in terms of development and usage costs, regarding both financial and resource usage aspects, for B3G networks. Service development platforms for B3G networks will be effective and successful only if the services they deliver are adaptive and offer quality of service guarantees to users despite the uncontrolled open wireless environment, which will be a key focus of the PLASTIC project.

3.2 The PLASTIC Platform

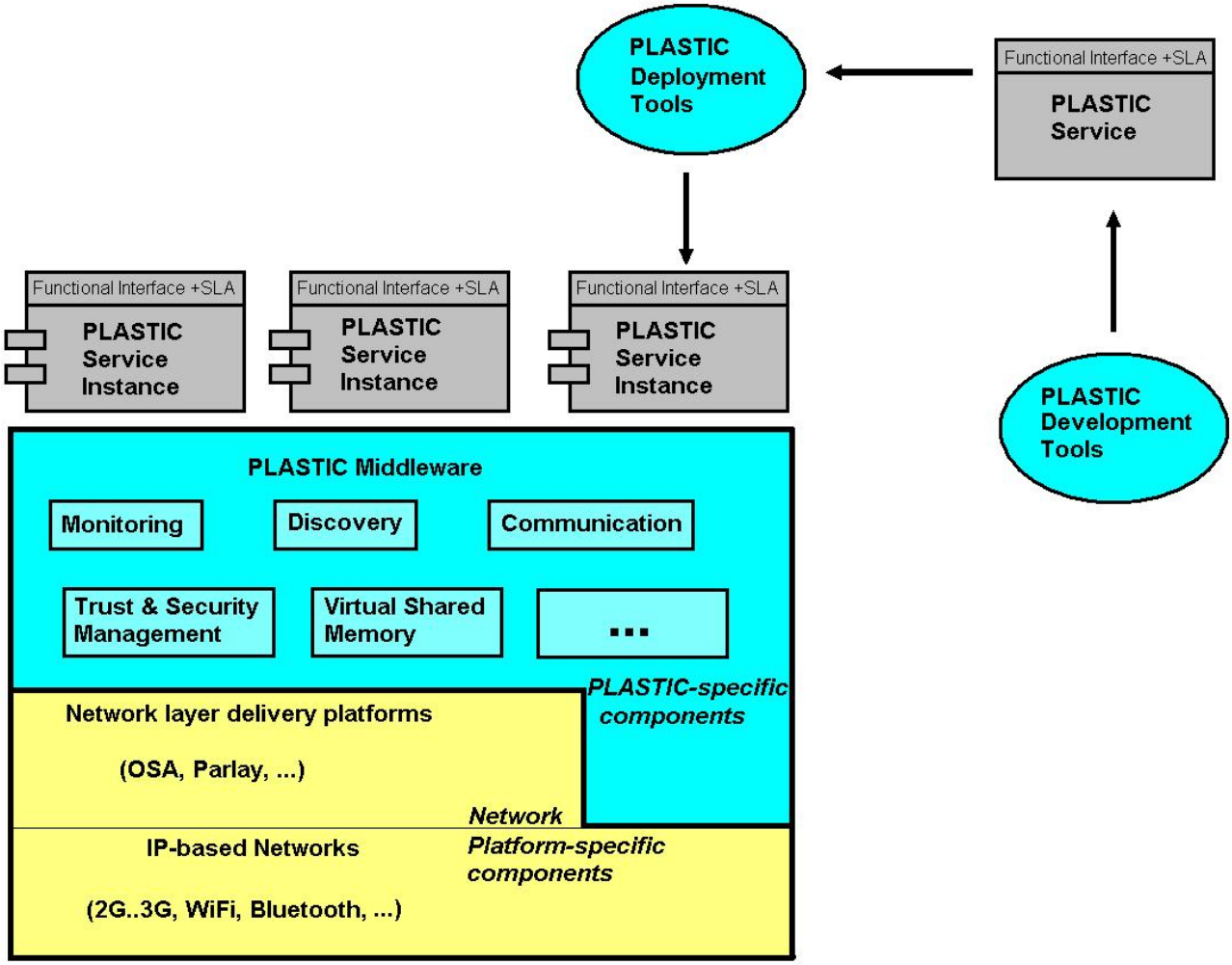


Figure 2: The PLASTIC platform

The PLASTIC platform (see Figure 2) will integrate software development methods and tools, and supporting middleware, enabling service provision in the open wireless environment. Specifically, the PLASTIC platform shall support:

- development of adaptive services for all-IP networks, i.e., development of QoS- and resource-aware, platform-independent, dependable services that adapt to the networking environment to deliver the best achievable quality of service and that may be deployed on a rich variety of devices, including wireless, resource-constrained devices;
- deployment and adaptive composition of services in the wireless environment, whether ad hoc, infrastructure-based or a combination of both, so as to realize complex and rich applications and make them available in most environments;
- run-time service management oriented toward monitoring and maintaining a quality of service that meets user expectations.

In order to fulfill the above requirements, research will be pursued in the following directions:

- Development and provisioning of robust adaptive services for the open wireless environment:
    - Service robustness will be promoted through integrated software engineering methods and tools, from design to validation. Service development will in particular build on the software engineering paradigms of service-oriented and component-based computing, i.e., an application is defined as the (possible) composition of autonomous, networked services, with an individual service being developed as a composition of components. This will allow the exploitation of existing development support oriented toward the functional robustness of applications.
    - Services will be adaptive to the environment with respect to resource availability and delivered quality of service, via a development paradigm based on Service Level Agreements (SLAs) and resource-aware programming.
- Middleware for service provisioning and composition in B3G networks:
    - The middleware will be service oriented, to enable integration and composition of heterogeneous software services, including services from both infrastructure-based and ad hoc networks.
    - The PLASTIC technology aims to be compatible with existing standards. PLASTIC will thus provide service developers with a set of platform-independent abstractions that will leverage, extend or interface with open APIs like those of OSA/PARLAY or of specific protocols. On the client side, the PLASTIC technology will allow service adaptation with respect to different access protocols (e.g., SMS, MMS, and WAP) and will provide the necessary run-time support to execute the service according to the user's expectation regarding QoS. In general, the PLASTIC middleware will allow interaction with, and exploitation of, services from the cellular network, using standard APIs defined for service platforms aimed at 2 to B3G networks.
    - The middleware will integrate key functions for supporting the management of adaptive services in the open wireless environment, dealing in particular with resource awareness, dependability, trust and security.
- Testing methods and tools to validate the dependability of mobile, adaptive services:

- Due to the mobility and strong dynamism of the considered systems, PLASTIC must provide new techniques for service evaluation and testing. In particular, dynamic adaptability is a pervasive requirement that is not adequately addressed by traditional testing methodologies. Hence, the PLASTIC platform will embody mechanisms to verify that an application, or part of it, will be able to "correctly" interact in different environments by taking advantage of the services that are available.
- QoS (particularly performance) constitutes another important characteristic of service-oriented architectures that needs to be carefully assessed. As QoS of mobile service-oriented applications is heavily influenced by communication mechanisms, the PLASTIC project will experiment with the applicability of testing methodologies for empirical QoS evaluation.

The two next sections further discuss the PLASTIC development environment and associated middleware, focusing on the requirements that we aim to address.

3.3 The PLASTIC Development Environment

Supporting the development of resource-aware and self-adapting components composing adaptable services requires focusing on the Quality of Service (QoS) properties offered by services. Although the functional properties of services are equally important for assuring the development of a competitive service, current component and service development technologies offer good solutions to achieve required levels of quality in the composition of components and services. To this respect, the PLASTIC platform will make use of consolidated technologies at the design and at the implementation level. In particular, the use of UML and MDA (Model Driven Architecture) for the design, and the use of the Java family languages for implementation will be considered.

For the management of non-functional/Quality of Service (QoS) properties offered by services, which is one of the innovative cornerstones of the PLASTIC platform, the work will rely on the specification of service level agreements (SLAs), i.e., abstract specification of the Quality of Service (QoS) properties offered by services as a compromise with the capabilities of the target platforms. In order to manage SLAs, we lay on analysis techniques that are capable of reasoning about the resources that components require in order to meet given service levels. An important objective is to devise the required notations and tools to support such designs. In order to support systematic development from the design level to the implementation of adaptable services and components, we will investigate the use of transformational approaches and rely on techniques and tools that emerge from MDA and generative programming research areas.

We will further develop a methodology for the validation of mobile adaptable component-based services from the standpoint of both functional and non-functional properties. The methodology will be supported by a test framework, part of the PLASTIC platform. Due to mobility requirement and to the strong dynamism of the considered systems, we need to study and implement new techniques for evaluation and testing. This includes the development of strategies to identify test cases aimed at validating interoperability aspects in the composition of networked services. These strategies will be built in line with the above innovative modeling and development approaches devised. In addition, the

same application will have to interact with newly dynamically added services at runtime. Hence we will have to devise related methods and tools to be deployed for on-line and off-line testing. We will also study the applicability of testing methodologies for empirical QoS evaluation.

3.4 The PLASTIC Middleware

The PLASTIC service-oriented middleware will support the PLASTIC development methodology for the deployment of mobile, adaptable services in beyond 3G networks. Specifically, the middleware shall be deployed over a large diversity of computing platforms, including wireless, resource-constrained devices, so that services may indeed be deployed on the mobile, wireless devices and not only accessed from them. The middleware shall further interface with the large diversity of networks composing the beyond 3G network. This includes benefiting from the functionality of latest network service platforms. Specifically, the PLASTIC middleware will be service-oriented, offering core middleware functions for the naming and discovery of networked services, and for interactions among networked services. The core middleware will further be enriched with a number of middleware-related services that are of prime importance in enabling mobile, adaptable services. Also, the middleware will allow applications to use high-level network-layer services –if and when available (e.g., location-awareness).

The core middleware supporting the PLASTIC mobile and adaptable services will build upon open standards related to service-oriented architectures and to network-layer service delivery platforms. Specifically, the PLASTIC middleware will adapt technologies related to the Web services architecture so as to enable development and deployment of PLASTIC-compliant Web services on various platforms, including wireless, resource constrained devices. Following the PLASTIC development environment for adaptable services, Web services will in particular be enriched with the specification of SLAs, further leading to related SLA-aware service discovery and access.

A key aspect of the PLASTIC middleware is to support interoperability among services deployed on devices that are heterogeneous in the software and hardware dimensions. Interoperability is addressed in PLASTIC by undertaking a service-oriented approach, which has been proven quite successful for the development of distributed applications in open environments, through the Web services architecture. PLASTIC further adopts an all-IP network view, leading to networks structured around IP domains. Services within one IP domain may then interact according to the access control policy of that domain, while interaction spanning multiple domains is made possible by dedicated bridges. The PLASTIC middleware operates over IP domains enabled by the network operator and/or ad hoc networking; PLASTIC does not offer any bridging functionality, for which various solutions may be found in the literature. The PLASTIC middleware makes available network-layer functionalities to the applications, like functions defined by service delivery platforms for 2 to 3G networks and more recently for beyond 3G networks. One key issue that arises is then which of the network-layer services should be offered to applications as is and which should be exploited at the middleware-layer and made transparent at the application-layer. For instance, the middleware may exploit the existence of multiple radio interfaces when interacting with the environment, so as to favor cost-effective interactions with respect to financial cost for the user, resource-usage and network-level quality of

service. Also, the PLASTIC middleware will rely on capabilities of the underlying network for seamless mobility. The impact of network heterogeneity on the middleware and application behavior will then be addressed through the elicitation of a comprehensive semantics for service interaction (i.e., connector types offered by the PLASTIC middleware), accounting for the various network-related properties of relevance (e.g., impact of mobility management on failure and synchronization semantics) based on adequate interfacing with the network.

Service management includes service discovery, access, reservation and accounting&billing, for which we additionally have to consider mobility and security issues. Regarding service mobility, we distinguish between logical and physical mobility. As mentioned above, physical mobility relies on its handling by the underlying network, and is addressed through the definition of appropriate interaction semantics. Logical mobility is supported by enabling the downloading of appropriate software components to dynamically compose services. Context-aware discovery of networked services in the PLASTIC middleware will take into account SLAs associated with the services, prior client-side subscriptions (service advertisement and service push) and reachable network domains. In addition, since there is a large number of service discovery protocols, each aimed at a specific network, we will reuse existing solutions to service discovery protocols interoperability (e.g., [15, 17]) so that the PLASTIC middleware may build upon them. Regarding service access, the PLASTIC middleware will rely on interaction functionalities provided by the core middleware and will further support functionalities for the orchestration/choreography of distributed services. Last but not least, the PLASTIC middleware will provide functions to enforce SLAs in the dynamic network environment. This includes monitoring of resources and adapting service composition and access based on specified SLAs and evolution of resource availability.

4 Conclusion

As information and communication technologies get increasingly pervasive, diverse and rich, *Software for the future* (Softure) must be adaptive to context, i.e., must change its behavior according to the context in which the software applications are provisioned and accessed. In addition, Softure needs to be self-adaptive according to both the resource- and user-centric context, which evolves over time. The IST project PLASTIC aims at offering comprehensive development support for Softure and focuses specifically on Softure to be deployed over next generation, B3G networks.

The PLASTIC methodology embraces both service creation and service execution. Thus, the PLASTIC platform comprises software tools for the creation, development and testing of software services/components and a middleware to support service deployment and execution with respect to established Quality of Service (QoS) parameters in the B3G networking environment. Service development also includes service/components discovery and their orchestration for the creation of new service instances. In this context, development is mixed with dynamic execution, which requires dynamic adaptation and customization of generic components and thus innovative validation techniques (e.g., pre-static certification, synthesis, etc.).

The PLASTIC approach is based on component and service technologies, making services adaptable with respect to the components they integrate and the networked services with which they interact, according to the networking environment and QoS (non-functional) requirements. Adaptable service composition, from the service- to component-level, is then realized through suitable characterization of components/services. This in particular allows validating components/services with respect to interaction and coordination, including QoS requirements, and independently of the actual execution platform.

The PLASTIC platform will support the development of diverse application services exploiting the significant number and diversity of wireless resources that are (expected to be) networked, thanks to next generation, B3G networks. "Killer applications" in such a pervasive mobile environment are yet to be identified. Also, success of the PLASTIC platform depends on the achieved cost-effectiveness, regarding the cost of both service development and service usage, compared to developing services for the infrastructure-based/cellular network. The PLASTIC project will address the latter issue, and may be the former one, through the development of a number of mobile e-services in the areas of eHealth, eVoting, eLearning and eBusiness.

References

[1] CACM. Issue on Aspect Oriented Programming. Communications of the ACM. 44(10). October 2001.

[2] C. Julien and G-C. Roman. Egocentric Context-aware Programming in Ad hoc Mobile Environments. In Proceedings of SIGSOFT FSE. 2002.

[3] M. Hicks, S. Weirich, and K. Crary. Safe and Flexible Dynamic Linking of Native Code. In Proceedings of TIC 2000, LNCS 2071. 2001

[4] M. W. Hicks, J. T. Moore, and S.Nettles. Dynamic Software Updating. In Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation. 2001.

[5] RAP Project. Resource Aware Programming. http://www.cs.rice.edu/~taha/RAP/.

[6] P. Inverardi, F. Mancinelli, and G. Marinelli. Adaptive Applications for Mobile Heterogeneous Devices. In Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops. 2002.

[7] P. Inverardi, F. Mancinelli, and M. Nesi. A Declarative Framework for Adaptable Applications in Heterogeneous Environments. Proceedings of the 19th ACM Symposium on Applied Computing, 2004

[8] D. Garlan. Software Architecture: A Roadmap. The Future of Software Engineering, aside ICSE00, ACM Press. 2000.

[9] S.-W. Cheng *et al*. Using Architectural Style as a Basis for Self-repair. In Proceedings of the 3rd Working IEEE/IFIP Conf. on Software Architecture (WICSA 2002). 2002.

[10] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. Towards Architecture-based Self-Healing Systems. In Proceedings of WOSS '02. 2002.

[11] B. Schmerl and D. Garlan. Exploiting Architectural Design Knowledge to Support Self-repairing Systems. In Proceedings of the 14th International SEKE Conference 2002.

[12] D. Garlan, S. Cheng, and B. Schmerl, Increasing System Dependability through

Architecture-based Self-repair, in Architecting Dependable Systems, R. de Lemos, C. Gacek, A. Romanovsky (Eds), Springer-Verlag, 2003.

[13] V. Issarny, C. Kloukinas, and A. Zarras. Systematic Aid for Developing Middleware Architectures. Communications of the ACM, Issue on Adaptive Middleware, 45(6). 2002.

[14] G. Blair, L. Blair, V. Issarny, P. Tuma, and A. Zarras. The Role of Software Architecture in Constraining Adaptation in Component-based Middleware Platforms. In Proceedings of the ACM/IFIP International Middleware Conference. 2000.

[15] P-G. Raverdy and V. Issarny. Context-aware Service Discovery in Heterogeneous Networks. In Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'2005). 2005.

[16] P. Grace, G. Blair, and S. Samuel. Middleware Awareness in Mobile Computing. In Proceedings of the 1st International ICDCS Workshop on Mobile Computing Middleware. 2003.

[17] Y-D. Bromberg and V. Issarny. INDISS: Interoperable Discovery System for Networked Services. In Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference. 2005.

[18] Proceedings of the International Conference on Trust Management. LNCS. 2003-2005.

[19] Sensor Networks. References at http://www.research.rutgers.edu/~mini/sensornetworks.html.

[20] E2R Workshop on Reconfigurable Mobile Systems and Networks Beyond 3G. http://e2r.motlabs.com/workshops/e2r-workshops. 2004.

[21] A. K. Dey and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-aware Applications. In Workshop on Software Engineering for Wearable and Pervasive Computing (CHI '99). 1999.

[22] C. D. Kidd, R. J. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner, and W. Newstetter. The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In Proceedings of the 2nd International Workshop on Cooperative Buildings.1999.

[23] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. IEEE TSE, 29(10). 2003.

[24] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service, ACM Transactions on Computer Systems, 19(3). 2001.

[25] C. Schuckmann, L. Kirchner, J. Schümmer, and J. Haake. Designing Object-oriented Synchronous Groupware with COAST. In Proceedings of the ACM Conference on Computer Supported Cooperative Work. 1996.

[26] N. A. Streitz, J. Geiÿler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, P. Seitz, and R. Steinmetz. i-Land: An Interactive Landscape for Creativity and Innovation. In Proceedings of the ACM Conference on Human Factors in Computing Systems. 1999.

[27] P. Tandler. Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. In Proceedings of UbiComp 2001: Ubiquitous Computing, LNCS 2201. 2001.

[28] A. Smailagic, P. Steenkiste, D. Garlan, and D. Siewiorek. Project Aura: Toward Distraction-

free Pervasive Computing. IEEE Pervasive Computing, 1(2). 2002.
[29] R. Cerqueira, A. Ranganathan, R. H. Campbell, M. Román, C. K. Hess, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. IEEE Pervasive Computing, 1(4). 2002.
[30] X. Fu, W. Shi, A. Akkerman and V. Karamcheti. CANS: A Composable, Adaptive Network Services Infrastructure. In Proceedings of the Usenix Symposium on Internet Technologies and Systems. 2001.
[31] V. Issarny, D. Sacchetti, F. Tartanoglu, F. Sailhan, R. Chibout, N. Levy, and A. Talamona. Developing Ambient Intelligence Systems: A Solution based on Web Services. Journal of Automated Software Engineering. Vol 12. 2005.
[32] JAIN. http://java.sun.com/products/jain/.
[33] 3GPP. http://www.3gpp.org/.
[34] PARLAY. http://www.parlay.org/.
[35] Ed. T. Zahariadis and B. Doshi. IEEE Wireless Communications Magazine. Special Issue on Applications and Services for the B3G/4G Era. October 2004.
[36] J2ME. http://java.sun.com/j2me/
[37] OSGi. http://www.osgi.org/.
[38] MOBIUS: http://mobius.inria.fr/
[39] H. Muccini, A. Bertolino, and P. Inverardi. Using Software Architecture for Code Testing. IEEE Transactions on Software Engineering, 30(3). 2004.
[40] V. Issarny and A. Zarras. Software architecture and Dependability. in Formal Methods for Software Architecture, M. Bernardo, P. Inverardi (Eds.), LNCS 2804, Springer-Verlag. 2003.
[41] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based Performance Prediction. Software Development: A Survey IEEE Transaction on Software Engineering. 2004.
[42] A. Bertolino and A. Polini. The Audition Framework for Testing Web Services Interoperability". In Proceedings of 31th Euromicro Conference on Software Engineering and advanced Applications, EUROMICRO 2005. 2005.

Requirements Composition and Refinement: Towards Composition-Centric Requirements Engineering

Ruzanna Chitchyan*, Awais Rashid*, Pete Sawyer*
* Computing Department, Lancaster University,
Lancaster, LA2 4YR, UK

Abstract. *Requirements represent what stakeholders are concerned about, i.e. their concerns. These concerns are often interrelated, and so are their representations in requirements or other artefacts. We discuss an approach to modularising and refinement of concern relationships and using them during requirements composition. Some concepts from Aspect-Oriented Software Development paradigm are used to support composition.*

Keywords: aspect-oriented requirements engineering, requirements composition, composition refinement

1. Introduction

Aspect-Oriented Software Development is a newly emerging software development paradigm, that comes with a promise to help in modularising crosscutting concerns, in their requirements, designs, and corresponding implementations.

Following the definition of [1], we define a concern as a matter of interest related to software development. Requirements represent the concerns at their analysis and specification phase. Concerns can also have design and implementation representations. A concern may be specified through one or more requirements, for instance, in some system, security could be a concern represented through such requirements as:

- Each user of the system shall be authenticated via login and password;
- Only an authenticated customer can purchase an item.

Thus, by definition, concerns are relative (as they vary depending on the stakeholders/developers perspectives); they are also dynamic, since they change with time and perspective. Concerns can participate in multiple relationships with each other. For instance, security can motivate an 'audit of records' concern, and hinder 'response time', etc.

The issue of crosscutting emerges when concerns are represented in some formalism and that chosen formalism does not support modular representation of certain concerns or their relationships. If a concern is not modularised in the software architecture, it has to be represented within modules that implement other concerns (in order to remain available in the

system). A well-known example of this is the problem of implementing logging in object-oriented systems.
Inter-concern relationships, on their part, propagate down into relationships between concern representations, and if not modularised adequately, often cause some representation of the related concerns to be included into the representation of the concerns themselves (e.g., inheritance relationship between objects is represented through injecting foreign key attributes into the objects represented in a relational database). This lack of modular representation of some concerns and their relationships causes scattering and tangling of concern representations in requirement, design, and implementation artefacts.
Scattering is the occurrence of units that belong to one concern in representations of others; and *tangling* is the contamination of concern representation with units alien to it [2]. Poor modularisation along with scattering and tangling, in turn, affects understandability, maintainability, and reusability of software artefacts.
The above discussion demonstrates the need for modular representation of concerns and their relationships. While concern modelling and modular representation have been addressed, for instance, in [1, 3], the issues related to concern relationships are relatively unexplored. Concern relationships and composition is the focus of our work.
In this paper we present our approach to modularising concern relationships and using them for requirement composition. The composition is used to detect and address conflicts between requirements early on, as well as gain an early understanding of relationships between artefacts of later development stages. Our approach to modular representation of relationships at requirements and later stages is based on encompassing these relationships in composition actions and operators. These actions and operators are then used for concern composition, preserving the semantics of concern relationships.
The concerns and their relationships are gradually refined into more detailed representations through an analysis process. This gradual increase of detail in requirements and actions/operators takes the requirements closer and closer to the design space, transparently bridging the gap between the two. Through this refinement, we also preserve the traceability of requirements [4, 5], thus contributing to improved requirements management.
In section 2 of this paper we discuss our composition-centric approach to requirement engineering, a simple example of this approach is presented in section 3. Section 4 presents related work, finally, conclusions and future work are discussed in section 5.

2. A Composition-Centric Approach (CoCA) to Requirements Engineering[1]

In this section we focus on main elements of the CoCA which are concerns, their relationships, representations, composition, and refinements. In CoCA (for requirements engineering phase) concerns are represented via requirements, and their relationships are

[1] Further work on CoCA (including work on operator semantics and derivation, etc.) is presented in [20, 21].

represented as actions and operators. The composition is supported through a refinable joinpoint model and refinable composition semantics - concepts brought in from the Aspect-Oriented Software Development paradigm. At each refinement level, the joinpoint model exposes structured points through which requirements can be composed and composition actions and operators provide clear semantics to the composition. Thus, composition is supported at different levels of refinement of concerns and their relationships.

2.1 Composition, Relationships, and Operators

One of the cornerstones of Software Engineering is the separation of concerns principle [6]. However, composition is an equally important issue as the end product of software development has to act as a single composed unit. Having a unified view of the system is desirable at all development stages, starting from requirements engineering. Thus, once the individual concerns, relationships and their corresponding requirements have been identified, it is beneficial to compose them into a single coherent requirements specification. This is useful not only for viewing the requirements as a whole to perceive the entire system, but also for detecting conflicts and inconsistencies between requirements early on and taking appropriate steps for their resolution before they have propagated further into the development process. Moreover, if, as desired, the modularity of requirement level concerns is preserved at the later stages of the software development lifecycle, the early composition of requirements will also reflect the relationships between corresponding artefacts at the later development phases. Hence, the relationships of artefacts of later lifecycle phases will be revealed very early on in the development process. This knowledge, in turn, will be valuable for early test planning, change management, etc.

Thus, it is important to capture the relationships and to propagate them down to design and implementation in a modular way. However, it is also apparent that relationships are relative and dynamic, as are concerns. It will be very difficult, if at all possible, to say that one concern always relates to another in a definite way, except in very general cases and in general terms. Each software system will have specific, often domain-defined, relationships between its concerns. That is why we propose relative and dynamic representation for concern relationships.

In this work relationships are represented thorough an open set of composition actions and operators. Rashid et. al [7] defined a set of composition operators and actions used in XML-based composition rules. We too follow similar representation. Some examples of composition operators and associated actions are presented in Figure 1 below, while examples of composition can be seen in section 3 of this paper.

Here operators reflect the nature of concern interrelationships, while actions define what should be done about these interrelationships. For instance, a temporal interval between two requirements (defined by between operator) can be merely observed without any intervention

(observe action) or intentionally enforced (enforce action). A further example of use of such actions and operators is provided in section 3.
Having an open set of operators and actions and an easy to use operator definition language, allows the analysts to adopt the pre-defined operators or provide new operators to serve their project-specific needs.

Fig. 1: Example of Actions and Operators

Action:
observe: monitor an additional condition over a set of requirements;
enforce: enforce an additional condition over a set of requirements;
provide: specify additional features to be incorporated for a set of requirements;
…

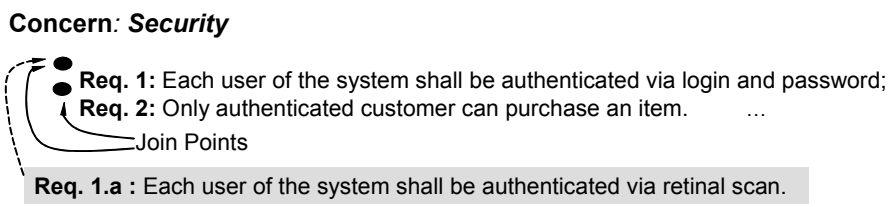
Operators:
between: temporal interval falling between satisfaction of two requirements.
for: additional features will complement requirements.
…

2.2 Join Points for Concern Composition Representation

Having flexible composition operators, however project-specific, is not enough, as relationships between concerns do not need to equally affect all elements representing a concern (e.g., all requirements representing a concern). Some representations can participate in a relationship fully, others partially or not at all. For instance, going back to our earlier example, the *purchase an item* concern is related to *security*, but buying can be represented as a requirement to pay for an item and a requirement to receive the item that has been paid for. While the requirement on paying for an item is closely related to the security concern for the on-line shop, the requirement on receiving the purchased item is not[2]. Thus, we need a mechanism for segregating the concern representations (and their sub-parts) that will participate in a certain relationship with other concern representations. This is where we draw on the power of Aspect-Oriented mechanisms such as joinpoints and pointcuts.
We characterise joinpoints as well defined points in the body of software modules. Since joinpoints are clearly defined, we can reference them and employ other software modules to interact with a given module at these points. For instance, a requirement in the concern representation is a clearly defined point in the body of a requirements specification document. We can use another requirement to replace or augment it, if we want.

[2] In another project we could have defined security to be also concerned with safe delivery of the purchased item, but here (as shown in the Introduction section) we only defined it as related to user authentication while purchasing an item at the on-line shop.

Fig. 2: Join points in the Security concern.



In Figure 2 above the authentication requirement Req. 1 in the *security* concern is being replaced with a new one Req. 1.a. Here we clearly defined the point (Req. 1 in security concern) in our artefact where we wanted to introduce a change, thus, the requirement statement in this case can act as *joinpoint*.
Pointcuts allow the referencing of more than one joinpoint at a time. For instance, we can say: replace all requirements within the security concern with some new ones. Thus, in this case, the reference "all requirements within the security concern" is our pointcut.
We can define the relationships between concerns in terms of actions and operators applied to selected parts of concern representations referenced through a pointcut.

2.3 Relationships and Pointcut Refinement

We have discussed so far that we aim to modularise the concern relationships as composition actions and operators, and we define joinpoints and pointcuts to segregate parts of concern representations which should participate in certain relationships with other concern representations. But we also recognise that concern representations are dynamic throughout the software development process. For example, requirements defined as statements (as in our example in Figure 2 above) may be derived from use-case analysis, they may be elaborated via details obtained during the course of analysis (using, for example, interaction diagrams) and so refined to augment understanding and expose the solution space (see example in section 3 below).
Once more detail is exposed in the requirements, we aim to make the concern relationships more precise using these new elements. This has prompted us to make both composition operators and pointcuts available for refinement in the same way as the requirements themselves. Thus, at the early stage of analysis, when only user-level requirements are available, we define composition using more general composition operators with reference to requirement statement level joinpoints, as shown in Figure 2. As analysis proceeds, leading eventually to the derivation of the more detailed software requirements, more precise composition operators, derived from the previous more generic ones, are used that reference finer-grain joinpoints, such as those of operation or attribute activation points (see example in section 3 below). Figure 3 below demonstrates some possible refinements for some operators.

Fig. 3: Refined *between* operator

Operator: ***between:*** temporal interval falling between satisfaction of two requirements.
Sub-Operators:
before**:** temporal interval falling between satisfaction of two requirements where the first requirement has been completed before the second one is commenced.
along: temporal interval falling between satisfaction of two requirements where the second requirement has commenced while the first one is still being satisfied.
…

Not only should composition operators be traceably mapped to artefacts of different granularity within the same lifecycle phase, but also between artefacts at the different levels of the lifecycle [8]. This however is not considered in the present paper.
An example demonstrating some of the issues discussed in this section is presented in section 3 below. It is also important to note that although the example demonstrates finer level joinpoints that match operation and attribute definition, refined join points can be used for matching any other refinements of previously more coarse-grained decomposition. We also believe that different decomposition approaches (e.g., per objects, features, roles, functions, etc.) may require different operators and actions and different types and granularities of join points. Our approach is not restricted to any decomposition or composition and allows all types of relationships to be reflected through corresponding composition operators and joinpoints. Thus, it is a multi-dimensional approach.

3. Demonstration of the Composition-Centric Approach

The case study used for the demonstration is an extension to an existing online shopping system. In this system the customers would have already been registered with the company and have credit accounts to buy on-line. The new extension is to provide bidding functionality. The registered customers will be able to place bids for items on sale. Only registered customers can participate in bidding and a bid should be accepted only if the customer has sufficient credit on his/her account.
The main concerns of this extension are *biding* functionality and the *security* of bidding. These two concerns can be represented in XML notation similar to [7, 9] via requirements, as demonstrated in Figure 4 below.

Fig. 4. Representation of Security and Bidding concerns.

```
<Concern name ="Security">
    <Requirement id ="1"> Authenticate customer </Requirement>
     <Requirement id ="2"> Authorize for bidding only authenticated customers </Requirement>
    <Requirement id ="3"> Ensure sufficient credit in account </Requirement>
</Concern>

<Concern name ="Bidding">
    <Requirement id ="1" > Place bid</Requirement>
</Concern>
```

The composition of the concerns presented in Figure 4 with action *observe* and operator *between* (similar to those in [9]) is presented in Figure 5 below. At this stage only the high-level requirements have been defined: the details of operations and data related to each requirement are not yet available. Consequently, the joinpoints available for composition are simply those of requirement statements within the concerns. This allows us to define only high-level pointcuts with reference to requirement statement joinpoints.
In Figure 5(a) we have defined a pointcut which contains two joinpoint sets. The joinpoint sets are the sets of points at which we expect the security and bidding concerns to interact. This additional element level is used to allow for grouping of affected concerns, which is particularly useful when requirements from more than one concern are affected by a group of requirements of some other concerns.
The rather high-level composition, demonstrated in Figure 5(b), states that some temporal interrelationship (defined by operator between) shall be observed between the JinpointSet 1 and JoinpointSet2 of sets of points in pointcut SecureBid. However, it is yet unclear what kind of temporal relationship it is, e.g., should the authentication requirement be activated first, followed by bidding, authorisation and credit check, or should some other order be selected?

Fig. 5. Composed concerns

```
<Pointcut name="SecureBid">
   <JoinpointSet id="1">
      <Concern name ="Security">
         <Requirement id ="1 2  3"/>
      </Concern>
   </JoinpointSet >
   < JoinpointSet id="2>
      <Concern name ="Bidding">
         <Requirement id ="1"/>
      </Concern>
   </JoinpointSet >
</Pointcut>
```

(a)

```
<Composition>
   <Action ="observe"/>
   <Operator ="between"/>
   <Pointcut name="SecureBid"/>
<Composition>
```

(b)

Upon carrying out additional analysis, using, for instance use cases, interaction diagrams, or any other requirements engineering methods, more detailed requirements will be acquired. In parallel with adding detail to the requirements (demonstrated in bold in Figure 6(a)), more detailed decisions can be made about requirement composition. Thus, if in our example we have used sequence diagrams, such details as operations and attributes associated to each requirement will be available, as well as the order of authentication, authorisation, credit check, and bidding will be defined. These details will in turn expose finer-grained joinpoints at which requirements can be composed, thus allowing more specific pointcuts, as shown in Figure 6(b).
The composition specification too becomes more precise as more requirement detail is exposed. In our example in Figure 6(c), we demonstrate how the composition initially specified through the operator *between* and action *observe* has been refined into the more specific *before* operator with more restrictive *enforce* action. This decision has been made

once the order of requirement activation has been established. Figure 6(c) contains references to two parts of the SecureBid pointcut. These are provided to match the semantics of the *before* operator: the first set of requirement joinpoints shall be activated before the second set.

Fig. 6. More detailed concerns and composition

```
<Concern name ="Security">
    <Requirement id ="1"> Authenticate customer if registered
        <attribute id ="1"> login</attribute>
        <attribute id ="2"> password</attribute>
        <operation id ="1"> checkLogin</operation>
        <operation id ="2"> checkPassword</operation>
    </Requirement>
    <Requirement id ="2"> Authorize for bidding only correctly
    authenticated customers
        <operation id ="1"> authorise</operation>
    </Requirement>
    <Requirement id ="3"> Ensure sufficient credit in account
        <attribute id ="1"> bidAmount</attribute>
        <operation id ="1"> checkPassword</operation>
    </Requirement>
</Concern>


<Concern name ="Bidding">
    <Requirement id ="1"> Place bid
        <attribute id ="1"> bidAmound</attribute>
        <operation id ="1"> placeBid</operation>
    </Requirement>
</Concern>
```

(a)

```
<Pointcut name="SecureBid">
  <JoinpointSet id="1">
    <Concern name ="Security">
      <Requirement id ="1">
        <operation id ="1 2"/>
      </Requirement>
      <Requirement id ="2">
        <operation id ="1 2"/>
      </Requirement>
      <Requirement id ="3">
        <operation id ="1"/>
      </Requirement>
    </Concern>
  </JoinpointSet >
  <JoinpointSet id="2">
    <Concern name ="Bidding">
      <Requirement id ="1">
        <operation id ="1"/>
      </Requirement>
    </Concern>
  </JoinpointSet >
</Pointcut>
```

(b)

```
<Composition>
  <Action ="enforce"/>
  <Operator ="before"/>
  <Pointcut name="SecureBid"
            joinpointSetId="1"/>
  <Pointcut name="SecureBid"
            joinpointSetId="2">
<Composition>
```

(c)

In this section we have demonstrated that requirements specification, pointcuts, and compositions can be defined in generic terms, independent of any particular requirements engineering approach, decomposition, or implementation language. We have also shown that as the requirements specification is refined towards design and implementation, pointcut and composition definitions, along with composition operators, can also undergo such a refinement.

4. Related Work

The work in [9] also addresses representation of concern relationships through composition. However, the composition rules in [9] are defined once per concern at the top concern level and are not refineable like our composition operators. Our work is also distinguished by the introduction of explicit notions of *refinable* requirement-level *joinpoints* and *pointcuts* and the *gradual refinable composition* of concerns and requirements.

Both [9] and our work use the idea of multidimensional separation of concerns put forward in [10, 11] in that the decomposition is not restricted to any single dimension. However, [10, 11]

do not deal with concern relationships at all; and unlike [9], our work provides improved support for traceability and mapping of concerns to later stages of the software development lifecycle.

Some similarity to our work in modelling concerns along with their relationships can also be found with the approaches proposed in [1, 3] and [12]. In this work Sutton et. al. argue for the need to model concerns as first class entities. They put forward a schema for concern classification where a section for relationships is also included. These relationships, however, are classified only into broad categories, such as Membership, Generalisation, etc. While we certainly value such classifications and use them for defining some classes of composition operators, the relationships in our work need not be simply categorical. We can also use more concrete, domain specific relationships, taking account of their relativity and flexibility in each specific case. We also use relationships as the basis for composition at different levels of granularity, while Sutton et. al. simply classify them.

The issue of relationships between concerns has also been discussed in more established requirements engineering approaches, such as the NFR Framework [13]. Here relationships are considered from the perspective of how some concern (a *softgoal*) contributes to another concern. The contributions are limited to such levels as strong positive, weak positive, strong negative, weak negative, etc. We reflect such relationships within composition operators as well as allowing for selective application of the operators to parts of concerns at different granularity.

Some work on requirement management also considers the dependencies between requirements and they affects on traceability [5], release planning [14], as well as change management [15], or importance of dependency types for particular development situations [16]; however none of these consider either composition of requirements, or relationship refinement.

Our work also expands some ideas from [7, 17]. In particular we use the idea of composition operators and their XML-based representation, as well as XML-based representation of requirements. However, in this work we depart from viewpoint-oriented requirement decomposition and embrace multidimensionality of requirements specification. Moreover, we expand the semantics of composition operators to act as encompassing units for concern relationship representations. We also add the notions of relationship traceability, composition refinement, as well as operator and concern granularity and joinpoint and pointcut granularity.

5. Conclusions and Future Work

In this paper we have presented the Composition-Centric Approach to Requirements Engineering. The approach supports requirement composition all the way through form the very initial high-level requirements identification stage up to detailed requirement specifications. Amongst the benefits of such composition support are:

- holistic view on the system requirements all through requirements engineering process;
- assistance with very early conflict and inconsistency detection, thus preventing potential costs of late corrections;
- exposure of relationships between requirements, thus assisting in requirements traceability, requirements reuse and software release planning;
- discovery of expected relationships between artefacts of later development stages early on, thus providing basis for such activities as software change management, test planning, etc.

Besides composition-centric focus, other key contributions of this approach are the notion of composition refinement and explicit representation of concern relationships. The relationships are explicitly represented though semi-formal operators, complemented with composition actions. The relationships, modelled as composition actins and operators, can be defined and adopted by the users for each specific development project. Yet, both relationships and composition specification are dynamic, as they can be evolved and refined as more detail about the requirements and relationships is revealed throughout the requirements engineering process. This flexibility of composition is based on the *refinable joinpoint model* and *refinable composition semantics*.

CoCA approach also provides a strong support for traceability, as it preserves concern identities linked to their representations, and modularity of concern relationships all throughout requirements engineering (and further on, if this approach is followed at later stages of development lifecycle).

As part of our future work we intend to further investigate the concern relationships and their corresponding composition operators that represent the relationships at different levels of granularity. We are also working on more precise pointcut definition and investigating the types of decomposition and their related joinpoint refinements. A larger case study demonstrating our approach is also under development. Yet another avenue of our research is looking at ways of identifying relationships between concerns and their requirement level representations. In this respect we are looking at linguistic engineering methods [18] developed in the REVERE project [19] as well as traditional requirements engineering approaches.

References

[1] S. M. Sutton and I. Rouvellou, "Concern Modeling for Aspect-Oriented Software Development," in *Aspect-Oriented Software Development*, R. E. Filman, T. Elrad, S. Clarke, and M. Aksit, Eds.: Addison-Wesley, 2004, pp. 479-505.

[2] W. Harrison and H. Ossher, "Subject-Oriented Programming - A Critique of Pure Objects," presented at Proc. 1993 Conf. Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 93), 1993.
[3] S. Sutton and I. Rouvellou, "Modeling of Software Concerns in Cosmos," in *Proc. 1st Int' Conf. on Aspect-Oriented Software Development (AOSD-2002)*, G. Kiczales, Ed., 2002, pp. 127-133.
[4] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," presented at International Conference on Requirements Engineering, Colorado Springs, Colorado, USA, 1994.
[5] B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability," *IEEE Transactions on Software Engineering*, vol. 37, 2001.
[6] E. W. Dijkstra, *Selected Writings on Computing: A Personal Perspective*: Springer-Verlag.
[7] A. Rashid, A. Moreira, and J. Araujo, "Modularisation and Composition of Aspectual Requirements," presented at 2nd International Conference on Aspect Oriented Software Development (AOSD), Boston, USA, 2003.
[8] S. Katz and A. Rashid, "From Aspectual Requirements to Proof Obligations for Aspect-Oriented Systems," presented at International Conference on Requirements Engineering (RE), Kyoto, Japan, 2004.
[9] A. Moreira, J. Araujo, and A. Rashid, "A Concern-Oriented Requirements Engineering Model," presented at Conference on Advanced Information Systems Engineering (CAiSE'05), Porto, Portugal, 2005.
[10] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. Sutton, "N Degrees of Separation: Multi-Dimensional Separation of Concerns," presented at Proc. 21st International Conference on Software Engineering (ICSE 1999), 1999.
[11] P. L. Tarr and H. Ossher, *Hyper/J user and Installation Manual*: IBM Research, 2000.
[12] W. Harrison, H. Ossher, S. Sutton, and P. Tarr, "Concern Modeling in the Concern Manipulation Environment," IBM Research Division Thomas J. Watson Research Center, Yorktown Heights, NY, USA RC23344 (W0409-136), 2004.
[13] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*: Kluwer Academic Publishers, 2000.
[14] J. Karlsson, S. Olsson, and K. Ryan, "Improved Practical Support for Large-scale Requirements Prioritisation," *Requirements Engineering Journal*, vol. 2, pp. 51-60, 1997.
[15] G. Kotonya and I. Sommerville, *Requirements Engineering - Processes and Techniques*: John Wiley & Sons, 1998.
[16] A. Dahlstedt and A. Persson, "Requirements Interdependencies - Moulding the State of Research into a Research Agenda," presented at The Ninth International Workshop on

Requirements Engineering: Foundation for Software Quality (REFSQ 2003), held in conjunction with CAiSE 2003, Velden, Austria, 2003.

[17] A. Rashid, P. Sawyer, A. Moreira, and J. Araujo, "Early Aspects: a Model for Aspect-Oriented Requirements Engineering," presented at International Conference on Requirements Engineering (RE), Essen, Germany, 2002.

[18] A. Sampaio, N. Loughran, A. Rashid, and P. Rayson, "Mining Aspects in Requirements," presented at Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005), Chicago, Illinois, USA, 2005.

[19] P. Sawyer, P. Rayson, and R. Garside, "REVERE: Support for Requirements Synthesis from Documents," *Information Systems Frontiers*, vol. 4, pp. 343-353, 2002.

[20] R. Chitchyan and A. Rashid, "Tracing Requirements Interdependency Semantics," to be presented at Workshop on Early Aspects (to be held with ASOD 06), Bonn, Germany, March 2006.

[21] R. Chitchyan, A. Sampaio, A. Rashid, P. Sawyer, S. Khan, “Initial Version of Aspect-Oriented Requirements Engineering Model”, Lancaster University, AOSD-Europe project report (D36) No: AOSD-Europe-ULANC-17, February 2006.

Business Modelling Environment BMETool

Miguel Montesdeoca* Jose Juan Hernández**
Ana Plácido, Mario Hernández ****
*MMCICOM Telecomunicaciones
**IUSIANI, University of Las Palmas de Gran Canaria

Abstract. *BMETool will provide a dynamic, interactive and interoperable modelling environment to the enterprises, through platforms where they can share the maintenance and operational costs.*

Keywords: Modelling, business, architecture, metadata

1. Introduction

1.1 Objectives

The main objective of BMETool project from a scientific point of view is to respond to the scientific-technical problem of how to achieve an adequate combination of business modelling at abstract organisation level and a middleware integration support based on services and service ontology. Initially, the aim is to integrate the service meta-models within the business meta-model.

From a more specific perspective, this main objective leads us to others, as listed below:

- Empirical Testing. To provide case studies of business modelling using available standards, and their subsequent evolution.
- To investigate dynamic business process analysis and modelling from the point of view of Digital Business Ecosystems.
- To design user interface design patterns for eBusiness from the perspective of optimising usability and processes in a comprehensive, rather than individualised manner.
- To study and validate the most suitable service modelling approach: with generic solutions at one extreme and more specific approach based on business type at the other.
- To validate the hypothesis that modelling in a business field with digital ecosystems may lead to a working solution for the business model approach to new technological mediums and their medium and long term feasibility.

- Identify and study the critical success factors to analyse business needs, and to ensure strategic alignment of Business Models with Operational and Technological configuration model

1.2 Inter-operability

The BMETool consortium will consider the kinds of inter-operability recently defined by the EIF (European Inter-operability Framework)[1]: organisational, semantic and technical inter-operability.
Organisational inter-operability deals with defining the business objectives, modelling business processes and the exchange of information between co-operating entities. It identifies and satisfies the requisites of the user community with the availability of user oriented accessible services.
Semantic inter-operability ensures that the meaning of the information exchanged is comprehensible for other applications. This semantics defined on XML vocabularies should be based on an in depth review of trends and the recent standardisation results.
Technical inter-operability covers the essential aspects to be able to combine and link services and information systems. The EIF distinguishes four levels of sophistication. The first two concern front office inter-operability (interfaces, presentation and exchange of data, etc.) and the other two, the back office (data integration, middleware, inter-connection services, etc.).

1.3 Business Modelling

Business models and methodologies are being effectively adopted by information systems engineer to describe important aspects of many organizations. The expressive power of these models is desirable not only to describe but also to design and deliver Web Applications and Web Services for these organizations. On the other hands, models and methodologies for implementing Web solutions are primarily focused on software engineering and technical aspects. This may fail in capturing important business aspects of the services to be delivered.
Also, the issue of interoperability between different businesses is becoming critical. In developing and modelling enterprise applications, it is necessary to integrate applications on both, operational and technological levels.
Recently, the Object Management Group introduced the Model-Driven Architecture (MDA) initiative as an approach to system-specification and interoperability based on the use of formal models. In MDA, platform-independent models (PIMs) are initially expressed in a platform-independent modelling language, such as UML. The platform-independent model is

[1] ICT Industry Recommendations - BRUSSELS, 18 FEBRUARY 2004)

subsequently translated to a platform-specific model (PSM) by mapping the PIM to some implementation language or platform (e.g., Java) using formal rules.
At the core of the MDA concept are a number of important OMG standards: The Unified Modelling Language (UML), Meta Object Facility (MOF), XML Metadata Interchange (XMI), and the Common Warehouse Metamodel (CWM). These standards define the core infrastructure of the MDA, and have greatly contributed to the current state-of-the-art of systems modelling. Other business modelling languages such as ARIS [17], IDEF0 [18], IDEF3, as well as languages based on CATALYST [CSC] notation have become very popular.
ARIS (Architecture of Integrated Information Systems) is a unique and internationally renowned method for optimising business processes and implementing application systems. The ARIS-approach provides a generic and well documented methodological framework. The ARIS-architecture distinguishes between organization, function, information and control view. In Scheer (1994) is argued that a formal language imposes restrictions on the day-to-day usability by potential end users. Business processes are described by process chain diagrams. ARIS focuses on the analysis and requirements definition phase during the design of managerial information systems, not on the execution of business processes. The modeling is done using a toolset instead of a language. The information captured by the ARIS toolset is stored into a database following the ERM (entity-relationship-model).
IDS Scheer recently announced a new of its ARIS EasySCOR solution that is based on the Supply-Chain Council's (SCC) Supply-Chain Operations Reference model (SCOR model). The SCOR model captures the council's consensus view of supply chain management. While much of the underlying content of the model has been used by practitioners for many years, the SCOR-model provides a unique framework that links business processes, metrics, best practices and technology features into a unified structure to support communication among supply chain partners and improves the effectiveness of supply chain management and related supply chain improvement activities.
The project team will also review other relevant approaches and achievements. For instance, the works of the University of St. Gallen, mainly those related to Business Model Design and Architectures for Business Networking.

1.4 Digital Business Ecosystems

Digital business ecosystems for SMEs is a research area that aim at providing an environment and suitable operative models enabling small-and medium-sized organisations to co-operate, through the idea of dynamic virtual organisations.

DBEs are the logical continuation of eBusiness, incorporating a completely new concept. The aim is to offer the additional functionality of an 'Evolution Environment', in which phenomena such as self-organisation and self-optimisation, which can be seen in the natural

world, are abstracted and transferred to a platform where they are available as services, with the aim of modelling the dynamic behaviour of organisations using these metaphors as frameworks of reference. This unique capacity will enable the ecosystem to suggest ongoing improvements to services and transactions as part of that same ecosystem.

In dynamic virtual organisations, it is required that all participating parties have a common understanding of the offering to be supplied. To achieve the understanding goal, an ontology that conceptualises and visualizes the ebusiness context is required. Ontologies provide concepts, relations between these, and rules which are supposed to be interpreted the same way by stakeholders, to conceptualise a specific domain.

The aim of eBusiness ontology is to create a shared, formal, and explicit conceptualisation of an eBusiness model:

- Conceptualisation refers immediately to business model. A conceptualisation is a model of reality, the business logic.
- Shared refers to idea that stakeholders should interpret a business model in the same way (ontological commitment); this is specifically important for e-business since many stakeholders from multiple enterprises are involved.
- Formal refers to a machine-understandable e-business model, such that software can support and analyze a business model. An eBusiness model should be explicit; that is not only in the minds of people, but written down.

As well, to achieve interoperability, the acquisition of web service semantic is required. This is a time consuming and complex task whose automation is desirable, as signalled by many researchers in this field. This problem can be addressed by building specific service ontologies (e.g. TicketBooking). Different general purpose ontologies, such as BMO, e3 value, Resource-Event-Agent (REA) Ontology (Geerts and McCarthy 1999) or the Service Ontology (Akkermans, Baida et al. 2004), could also solve the interoperability goal.

1.5 Usability and eBusiness

Usability is becoming the object of increased interest within the software development world, as a key quality factor. Larman [1] sustains that there is probably no other technique with a greater disparity between its importance for the success of software development and its lack of attention and formal education than usability engineering and the design of user interfaces. Organisations are beginning to include usability requisites in their software specifications, since they are now aware of how a software product's usability level can influence their employees' productivity.

Most of the available models for usability design seem to be too narrow and focus too much on a general landscape of interaction problems, without appropriately and sufficiently relating it to the business context.

Software usability can benefit from knowledge of the user and their tasks. The user interface should be regarded as a crucial part of doing business and its design should therefore be based on business modelling and, at the same time, business processes should be designed with usability in mind.

When designing the user interface it is common to consider different usability factors. Effective use of contextual data about the users and their tasks is crucial for the design of usable and useful systems. Also, the knowledge about what works and what is not important, especially in projects where the time is limited but a high quality interface is still desired.

1.6 Model Driven Architectures

The proper quality management which all organisations now need to develop or sustain competitiveness and further their business activities is the most important challenge currently facing those responsible for implementing information technology in companies. Traditional business management models have become obsolete and are now being replaced by process-based rather than product-based management systems. Consequently, the architectural approaches selected should support rather than hinder business processes.

Model-driven architectures (MDA) represent a new paradigm in software development, where the models guide all the development process. This is named Model Engineering or Model Driven Engineering. MDA is a Registered Trade Mark of the Object Management Group.

Currently, software construction is undergoing constant change as regards implementation technologies, which in turn means that concerted efforts have to be made in both the design of the application, in order to integrate the different technologies involved, and in its maintenance, in order to adapt it to changes in requisites and implementation technologies.

MDA separate the business or application logic from the underlying platform technology and represent this logic with precise semantic models. MDA reinforce the use of an enterprise architecture strategy and to enhance the efficiency of software development. MDA provides a solution to business and technological changes, by allowing the construction of platform independent applications, which can be implemented in CORBA, J2EE or using Web Services.

In order to gain these benefits, MDA uses the following development process: the requisites are used to obtain a platform-independent model (PIM); this model is then transformed, with the help of tools, into one or more platform-specific models (PSM); and finally, each PSM is transformed into a code. Therefore, MDA incorporates the idea of transformations between models (PIM to PSM, PSM to code), which is why it needs tools to automate this task. These transformation tools are, in fact, one of the basic elements of MDA.

1.7 Enterprise Service Bus

The integration of applications is increasingly becoming a strategic factor which must be taken into consideration by companies. The integration of applications and middleware opens up new possibilities for SMEs, given that someone has to supply the functionalities that provide a high level of integration. This whole issue is gaining speed with the adoption of Web services that enable users to construct new applications extremely quickly, thereby taking advantage of existing products.

In the architecture that is going to be developed in this work package, we will deal with problems relating to the integration of business processes linked to the distributed control of business processes.

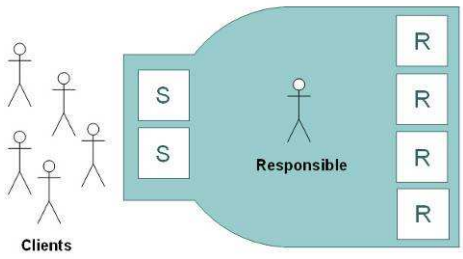
The system to be developed will integrate the concept of an ESB, or Enterprise Service Bus. An ESB is a type of middleware designed to respond to the integration needs of companies based on Service Oriented Architectures (SOA). In an ESB, all connected systems are either providers or consumers of services, even with legacy systems that do not provide services. Therefore, service oriented architectures (SOAs) can be deployed without demanding that participating systems offer a service-based interface.

2. Approaches

2.1 User Driven Approach

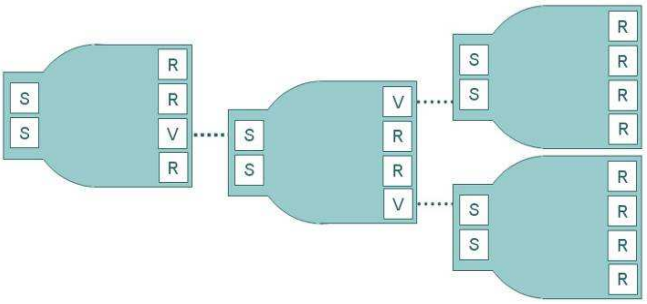
BMEtool will be designed around the effectiveness of the satisfaction of the user necessities, so the project is strongly user driven designed and includes tasks to ensure it from the beginning, through the logical building process in the definition genesis.
Starting from the user needs on secure, transparent, powerful and simple access, the functional and technical analysis will define a first step system metaphor by refining the conceptual model presented in this document.
The project includes activities to carry out a global check on integration and on transparency requirements which will make possible a progressive definition (refinement) of the modeling environment and the associated tools..
The logic is always user driven because the main target is to realize a wide accessible secure infrastructure where complexity is totally hidden for user. Finally a block that deals with a global overview on consistency and reliability has to be inserted.
This logical track end with a feedback on user requirements because the process can put in evidence new possibilities, so that the user driven logical process can be refined and go to further steps and help to define the final scenario.

2.2 Modelling Approach



A Business Unit in BMEtool

The modelling approach is based on Business units. They are atomic business entities promoted by an organisation (legal or virtual) and focussed on providing its clients with services. Any business unit is managed by a head whose task is to manage resources properly [R] to provide clients with services [S].



Business Units working together in BMEtool

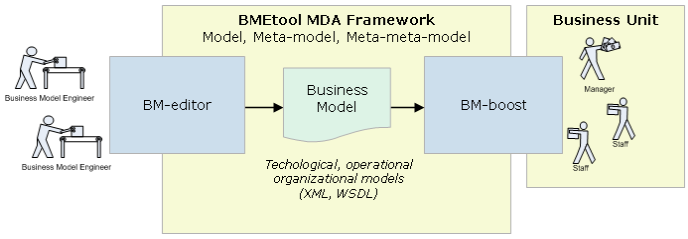
Modelling will be done at three levels: Organisational Operational and Technological.
As well, BMEtool will provide functions that will enable to couple business units creating new compound business units, making their composition and their internal assignations transparent to the outside.
BMEtool will include federation mechanisms for encapsulating the relations between business units that will allow transparent communication without the need of translation mechanisms.

3. Architectural Process

The initial concept of the architecture is inspired in two technological approaches: Model Driven Architecture and Enterprise Service Bus where the modelling approach is based in three abstraction levels: organisational, operational and technological However, requirements for specific MDA and ESB capabilities will need to be analysed in detail in order to identify suitable solutions patterns and implementation technologies. Solution patterns for ESB

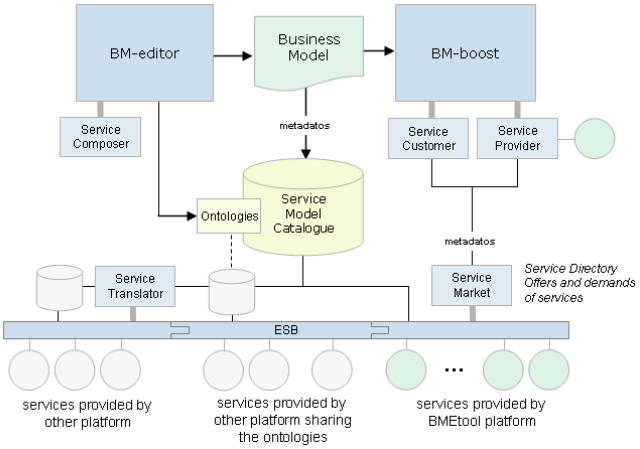
implementation such as Basic Adaptors, Service Gateway, Web services-compliant Broker, EAI Infrastructure for SOA, Service Choreographer, will be analysed [1].

The principal components that make up the architecture are, on the one hand, applications for editing and exploiting models, BM-editor and BM-boost, respectively; and on the other, the business models whose syntax and semantics are supported by modelling languages and meta-models. (Figure below) Users of the model editor (BM-editor) will be mainly business model engineers. This tool will help engineers translate the model on the basis of the requisites analysed in the business units. BM-boost, the exploitation tool, is mainly designed to be used by the technical and management staff of the business units, and aims to serve the business unit as a tool for ensuring the efficient execution of their everyday processes.



Architecture vision: BMEtool MDA Framework

Given that the architecture is oriented towards services, BM-editor will incorporate a service model management component (Service Composer). These models will be described using the appropriate standards and will be stored in a catalogue in order to enable their storage and subsequent consultation and discovery during the preparation of business models.



Architecture vision: BMEtool ESB

Learning form the achievements of the DBE project (Digital Business Ecosystems), the Service Composer will provide a **Service Recommendation Function** that will act as autonomous processes that manage organisation (Companies, SMEs, etc.) preferences, (either

business preferences or service preferences) and matches theses preferences with available business descriptions and service descriptions.
Before starting the design and implementation of the Recommendation mechanisms, the project team will study the existing business and service ontologies that capture the semantics of business models and service descriptions. These ontologies will be used to define the corresponding preferences for businesses and services. These preferences are attached to the business model of each organisation and will be used to generate the Configuration Model. In the Service Model Catalogue, the ontologies will be described in an XML based language (XMI) or OWL. A decision on this issue will be taken as soon as possible.
The **Service Model Catalogues** stores service descriptions in a widely accepted standard like Web Service Definition Language (WSDL). Business models should also be comprehensive enough to contain semantics for importing web services definitions from WSDL documents and exporting business process flows to an appropriate output standard like Business Process Execution Language for Web Services (BPEL). Internally, the BPM modelling process follows four phases: Service requirements; Service Import; Workflow Composition; and Workflow Export. Thus, internal components are required to discover external services and to publish services to other business units. BM-boost would have two associated components for managing services: Service Customer, whose function would be to locate, discover, negotiate and exploit the services provided by other business units; and Service Provider, whose mission it would be to provide services to other business units and publish and sell them on the service market.
The external available services (mainly those described in a different way) will be "captured" and described by the "external services tool". A specific component identifies an external service and its characteristics. These tools can present two different modes: The simple one, where it just browses for Services and the advanced one where it provides guidance for query formulation when a user looks for particular characteristics.
From the perspective of service access, we need to assess solutions that will enable the system to remain heterogeneous and approach that of a Digital Business Ecosystem, in which resources (software, servers, databases, etc.) are combined with no strict, pre-defined hierarchy.
We will therefore have catalogues of models distributed among different servers or platforms. Some models will be stored in system databases made up in reality by the merging of different individual repositories. Other models will be stored in the services themselves (for example SDE or Service Data Elements in the OGSI service model), or even through HTML tags embedded into websites.
Since some models will be located in external repositories, in order to ensure their proper management, we will need to resolve heterogeneity in a distributed environment. Furthermore, we will need tools that enable the creation of repositories at a certain level through the injection of one or more metadata distributed at a lower level.

BMEtool will integrate the concept of an ESB, or Enterprise Service Bus. The purpose of an ESB is to provide a transparent medium for exploiting services, in which the service customer is abstracted from the technology and specific implementation with which it was developed. For example, a merger mechanism may exist to encapsulate the relations between business units that enable a transparent relationship without needing to continuously identify the service customer.
The ESB could contain a mapping component, Service Translator, which will be used to ensure the transformation from different Ontology languages to the standard Ontology descriptions used in the BMEtool.

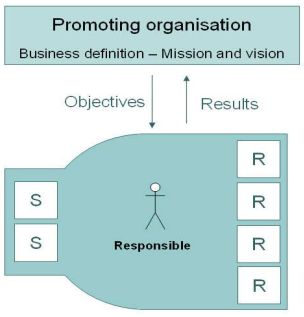
4. Interoperability Mechanisms: Models and Ontologies

The BMEtool approach is based on metamodelling concepts as well but with different abstraction levels. Three modelling levels have been identified:

1. Organisational. From a strategic point of view, business models reflect the need of targeting the market, defining the business goals, monitoring these goals and assigning resources.
2. Operational. From a tactic point of view, business models express how companies provide their clients with services and how they manage their resources, processes, customer relations, etc.
3. Technological. From a technological point view, an IT infrastructure is needed to provide support for business everyday tasks.

4.1 Modelling at organisation level

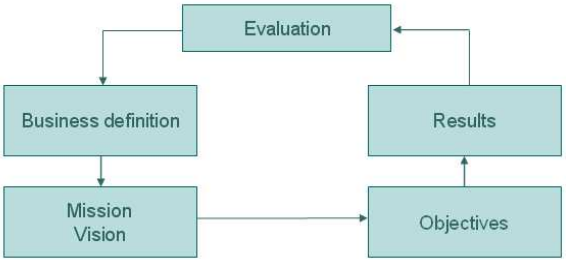
"Re-engineering business" is a response to the problems caused to companies by competitiveness in today's changing world. Internet introduces a substantial change in the way business is done. It is not enough to introduce technology; SMEs have to make an effort to restructure their business.



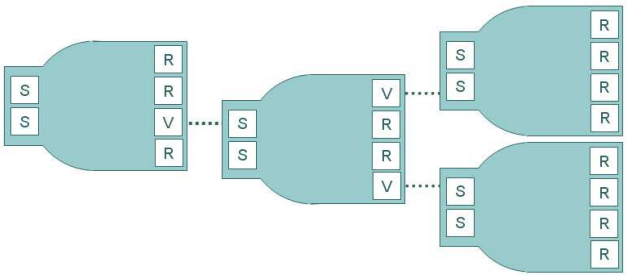
The reconstruction of processes requires companies to break away from the old processes of the division and specialisation of tasks. The starting point is to focus companies on the "processes", that is, the activities that lead to valuable results for the client.
Our approach to model at this level of abstraction is using the concept of Business Unit. A businesses unit is an atomic business entity promoted by an organisation (legal or virtual) and focussed on providing its clients with services. Any business unit is managed by a head whose task is to manage resources properly [R] to provide

clients with services [S].
At this level, it is required then definition of the strategy of a business unit. Strategic planning is a continuous process that considers the nature of the business, long term objectives are defined (mission-vision), quantifiable objectives are identified and strategies are developed to attain these objectives, the head of the business unit is appointed and suitable resources are assigned to meet the objectives.
The typical tasks on the strategic level are:

1. Create a shared vision of the business, establishing how the company should compete (clients and markets, products and services)
2. Define the criteria that will govern the business purpose; and evaluate the results
3. Evaluation of the results is necessary for monitoring and continual follow up of the business.
4. Design the operation model that optimises the use of resources and the quality of the service (QoS) provided to clients.



At this level, it is required then definition of the strategy of a business unit. Strategic planning is a continuous process that considers the nature of the business, long term objectives are defined (mission-vision), quantifiable objectives are identified and strategies are developed to attain these objectives, the head of the business unit is appointed and suitable resources are assigned to meet the objectives.
Small firms are face by a changing culture in which they must participate in clusters or networks to guarantee their survival in the new knowledge economy. The action of establishing relations that shape these groups is known as networking. Business networking creates a space where ideas are exchanged and initiatives are pooled in order to carry out collective actions. Organisations can be more flexible by dynamically assigning virtual resources (based on the services provided by other organisations), thus contributing to the development of Digital Business Ecosystems.

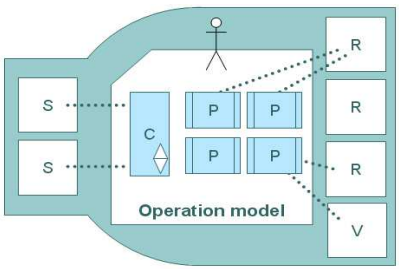


Business Units working together in BMEtool

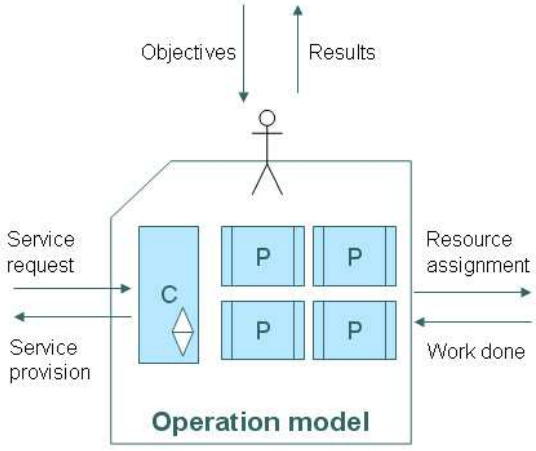
Organisations can be more flexible by dynamically assigning resources (based on the services provided by other organisations), thus contributing to the development of Digital Business Ecosystems. BMEtool will provide tools that will enable groupings of business units to define themselves as new compound business units or virtual business units, making their composition and their internal assignations transparent to the outside.

4.2 Modelling at operational level

Any business unit will have to develop an operation model to provide its clients with services, by managing its resources optimally. The manager is the maximum responsible of coordinating resources to provide services to the customers. The idea behind this project considers that:



- Client orders are planned and organised in the service queue [C]
- Processes and tasks are generated [P] for providing a service.
- Tasks are assigned to resources automatically or manually to implement processes.
- The head supervises the execution of the processes.



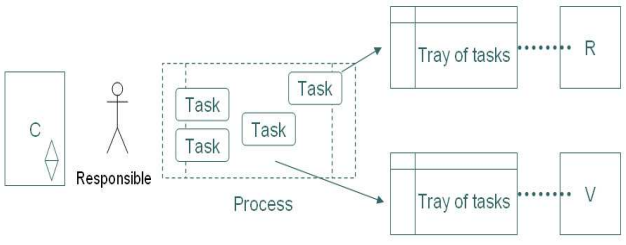
The main tasks on an operational level are as follows:

1. Monitor fulfilment of objectives
2. Analyse clients' requests for services
3. Define the work flow
4. Assign tasks to resources (dispatching)
5. Evaluate that resources are operative and carrying out tasks properly.
6. Monitor the execution of tasks
7. Evaluate the quality of the service
8. Provide measurements of the results

Communication is a horizontal set of functions in any company. They are absolutely essential for their operational functioning and contribute to building identity and culture. Communication could be either internal to the business unit or external with other business units. The goal is that BMEtool gives support to those functions by means of the following functions:

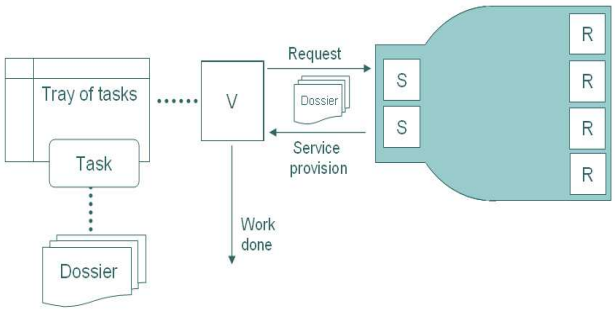
- Authentication. The availability of solid authentication functions provides users with a secure platform to develop collaboration, communication, trade, and business processes.
- Transmission. The development of channels for good communication will affect the perceptions that the workers and the environment have of the company. Communication helps to create commitment and cohesion among workers.
- Coordination. Coordination is basic to generating joint work strategies and strengthening the development of common goals.

The operational model will be supported on these functions based in the following approach: The head (responsible) deals with all the outstanding service requests. To provide the service, he generates tasks that he assigns to resources of the business unit. The resources (real and virtual) of a business unit have an associated tray where the tasks they have to do are placed.



Business unit task dispatcher

Each task has an associated dossier containing all the necessary documentation for carrying out the task. The resources need to have access to all or part of this documentation



Interoperating business units

4.3 Modelling at technological level

IT's are an integral part of eBusiness, and so the design and construction of a reliable infrastructure is considered a key aspect of business management. Therefore, if firms are to have success with eBusiness, they must create an optimized IT infrastructure to respond to business requirements.

In general, an eBusiness infrastructure comprises the following components: hardware, operating systems, networks, security, application servers and managers for content, applications and services. The infrastructure must be complemented by procedures and personnel to put it into operation and maintain it in order to guarantee the necessary levels of service.

An infrastructure for eBusiness requires transparent integration of all its constituent services with the following criteria for quality:

- Flexibility. The adoption of eBusiness is an evolutionary process that begins with simple implementations that become more complex as the business model becomes increasingly integrated with IT.
- Scalability. To meet unforeseen variations in customer demand and user workload, an eBusiness infrastructure must be scalable to adapt to variations in the workload while maintaining a high level of availability.
- Reliability. To guarantee the stability, continuous and secure operation and availability of the applications by minimising the degradation effects of all contingencies on the functioning of the system.
- Interoperability. To resolve relations of trust between business units and problems of communications between business units. There are three possible solutions to the problems of communication: addressing communication problems individually, discovering the services that other companies offer automatically and federation of services around a common ontology.

SME's do not have the possibility of building such an infrastructure individually. Firstly, they do not have the resources and, secondly, because investment in IT must be related to their contribution to the organisation's goals. SME's must perform the seemingly impossible task of offering quality IT services while controlling investment.

The fundamental vision of project is to link consumers and providers in such a way that the provision of services can be modelled on the provision of other utilities, such as water, electricity and telephone. For example, when consumers connect domestic appliances to the power supply, they do not have to worry about where the power station is located, or whether they have standard sockets for the plugs. The user does not need an electricity generator in the house, and all aspects of where the current is generated and how it arrives at the socket remain a mystery to the user, who simply makes use of the electricity service.

The intention is to solve the problems faced by organisations and **permit the dynamic configuration of the IT infrastructure** in line with organisational needs. While the organisation's own resources physically exist within the organisation, which must, therefore, be responsible for configuring, managing and updating them, an external resource exists virtually in the organisation by means of a service offered by another organisation. Ideally, an organisation could configure its entire IT infrastructure from external resources provided by many different organisations.

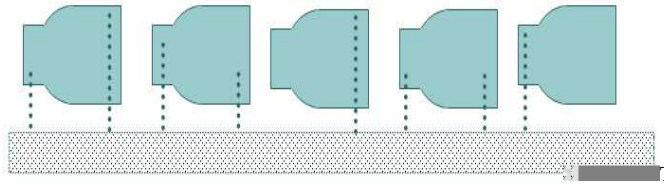
The management of eBusiness applications is complicated since companies must continuously update applications with new functionalities. Companies cannot afford to spend a year or more implementing the required management environment; on the contrary, management needs must be satisfied as soon as the application appears. Therefore, it is essential to be able to guarantee the immediate availability of eBusiness applications.

The orientation to services on the BMEtool guarantees that companies can dynamically change the eBusiness application by automatically migrating data. IT services are key services that can be provided by several business units.

Our approach integrates the management of the meta-data associated with services with meta-data associated with models, as the services meet the requisites of organisations expressed in the form of operational models built from meta-models that define the languages for creating the models. The operational models are requested in set up models that use the components available for meeting the requisites presented in the operational model as closely as possible. The evolution and improvement of the service platforms will evolve in line with the evaluation and improvement of the models requested to gradually approach the operational models of the organisations.

To automatically identify inter-operable services and select the best one for each user profile with a minimum human intervention, high level services must use the information on resource capacities and on their mechanisms for providing the service. Services must be characterised

by means of information that should include a description of their capacities and the policies for using them.



The interoperable approach will be based on federations. A federation is a mechanism for encapsulating the relations between business units that allows transparent communication without the need for translation. Federations not only solve problems of communications between business units. Identities can also be federated and, thus manage relations of trust.

4.3 Modelling language definition

One of the main tasks is to define the restrictions and characteristics of the modelling language and identify its elements: semantic and grammatical. The following aspects will be taken into account:

1. The modelling language must serve to specify, visualise, build-up and document the elements of a business ecosystem.
2. One of the objectives is to understand, design, mange and control the business ecosystems' information.
3. The language must capture the information regarding the static structure and the dynamic behaviour of the organisation.
4. Modelling will be done at three levels: organisational, operational and technological. Each organisation will be modelled as a discreet object collection that will interact to carry out a work for the benefit of the organisation.
5. The modelling language will unify the past experience on modelling and will incorporate best practices as a standard approach.
6. The language architecture must accomplish with the OMG´s Meta Object Facility specification (Meta-metamodel, Metamodel, Model and User´s objects).
7. The modelling language will be focued to the System Engineer or/and Business Engineer, who will use the language as a modelling tool to develop Business modelling environments.
8. BMEtool will integrate code generators to allow the models developed using this language will be executable in the selected business platforms.

5. Scoping and Integration in Existing Platforms

BMEtool goes further than simple organisation modelling, since this organisational model serves as the springboard for the automatic or assisted generation of the technological model required by the organisation. The integration of BMEtool in some of the existing eBusiness platforms selected in the project is a basic requisite for testing integration between different business units. The eBusiness platforms themselves are, at heart, business units that provide technological infrastructure services. One of these platforms is Dias.net, whose first version was the result of a 5th European R+D Framework Programme project. The platform itself will be defined as a business unit, providing the "Internet Spaces" service, among others. This Business Unit makes it possible to implement the most complex case, as it will be made up of the aggregation of the primary business units. Along with Dias.net, two other platforms will be integrated as business units. The final objective is to create a federation of business units.

Bibliography

[1] The OWL-S Services Coalition. OWL-S: Semantic Markup forWeb Services. White Paper. www.daml.org/services/owl-s/1.0/owl-s.pdf, 2003.
[2] C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating Experiments Using Semantic Data on a Bioinformatics Grid. IEEE Intelligent Systems, 19(1):48–55, 2004.
[3] ECOOP '2000 Workshop on Metadata and Active Object-Models, June, 2000, Cannes, France. Lecture Notes in Computer Science 1852, Springer-Verlag, Heidelberg. http://www.adaptiveobjectmodel.com/ECOOP2000/.
[4] Object Management Group, The Common Warehouse Metamodel (specifications, papers, presentations, OMG press kit, etc.). http://www.cwmforum.org/, http://www.omg.org/.
[5] D'Souza, D., "Model-Driven Architecture and Integration: Opportunities and Challenges", Version 1.1.http://www.catalysis.org/publications/papers/2001-mda-reqs-desmond-6.pdf
[6] OMG Model-Driven Architecture Home Page:http://www.omg.org/mda/index.htm
[7] OMG Meta Object Facility Specification, Version 1.3, September, 1999. http://www.dstc.edu.au/Research/Projects/MOF/rtf/. http://www.omg.org/.
[8] Object Management Group, XML Metadata Interchange Specification, Version 1.1, http://www.omg.org/.
[1] Larman, C.: Applying UML and Patterns. Prentice-Hall (2002)
[2] International Organisation for Standardisation: ISO 9241 - Ergonomic Requirements for Office Work with Visual Display Terminals (1995)
[3] C. Alexander, S. Ishikawa and M. Silverstein. A Pattern Language, volume 2 of Center for Environmental Structure Series. Oxford University Press, New York, NY, 1977.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.
[5] P. Coad, D. North, and M. Maryfield. Object Models: Strategies, Patterns, and Application. Yourdon Press, New Jersey, NJ, 2nd edition, 1997.
[6] J. Vlissides. Pattern Hatching: Design Patterns Applied. Software Patterns Series. Addison-Wesley, New York, 1998.
[7] D. C. Hay. Data Model Patterns: Conventions of Thought. Dorset House Publishing, New York, 1996.
[8] R. Robinson.. Understand Enterprise Service Bus scenarios and solutions in Service-Oriented Architecture. IBM. http://www.ibm.com/developerworks/ webservices/library/ ws-esbscen2.html.
[9] David A. Chappell Enterprise Service Bus. O'Reilly. 2004
[10] A. Kleppe, J. Warmer, and W. Bast, MDA Explained: The Model Driven Architecture Practice and Promise. Addison Wesley, 2003.
[11] A. Brown. An Introduction to Model Driven Architecture. Part 1: MDA and today's systems. IBM. The Rational Edge. 2004.
[12] D. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley Press, 2003.
[13] Richard Soley and OMG Staff Strategy Group, "Model Driven Architecture." November 2000.
[14] P. Harman, "MDA: An Idea Whose Time Has Come." Cutter Consortium, 2003.
[15] B. Selic, "The Pragmatics of Model-Driven Development," IEEE Software, Vol. 20, #5, September 2003.
[16] T. Gardner, C. Griffin, J. Koehler, and R. Hauser, "A review of OMG MOF 2.0 Query/View/Transformation Submissions and Recommendations Towards the Final Standard". IBM Whitepaper submitted to the OMG, September 17, 2003.
[17] D.K. Barry, Web Services and Service Oriented Architectures. Morgan Kaufman, 2003.
[18] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. Addison Wesley, 2001.
[19] CSC Catalyst Methodology, CSC Inc, 1995 Comparing two Business Model Ontologies for Designing e-Business Models and Value Constellations
[20] A. Thomas Manes, Web Services: A Manager's Guide. Addison Wesley, 2003.
[21] S. Mellor et al., MDA Distilled. Forthcoming from Addison Wesley, 2004.
[22] S. Mellor et al., Executable UML: A Foundation for MDA. Addison Wesley, 2003.
[23] J. Warmer and A. Kleppe, The Object Constraint Language: Getting Your Models Ready for MDA, second edition. Addison Wesley, 2003.
[24] J. Daniels, "Modelling with a Sense of Purpose." IEEE Software, pp8-10, Jan/Feb 2002.
[25] A.W. Scheer. Aris-Business Process Modeling, Springer Verlag. 1999

[26] S.C. Hill. 1994 A Concise Guide to the Idef0 Technique: A Practical Approach to Business Process Reengineering, Enterprise Technology Concepts Inc.

A European Open-Source Project Information Server

Chris Chedgey, Mícheál Ó Foghlú, and Eamonn de Leastar
Telecommunications Systems and Software Group (TSSG),
Waterford Institute of Technology

Abstract. *We propose an extensible Open-Source software project to collect data from the source code and plethora of tools used in development environments and to provide aggregated, filtered, analyzed, trended information to developers and managers.*

Keywords: software development, metrics, project management, open source

Overview

Each year, over €500B is spent globally to fund development teams that design, build, enhance and maintain software code. Despite this huge investment, software development is plagued by a lack of integrated, timely information for developers and management. This fundamental lack of visibility means that software development has remained as a largely inefficient and unpredictable activity, with significant implications for cost and efficiency.

Solving this problem is extremely challenging because to date there has been no unifying Information Model for so doing. We propose the development of a "Software Mapping Server" to automatically extract key data from the source code and plethora of tools used in development environments. This project map is then used as a source of aggregated, analysed, trended, filtered information for the creation of accurate and timely web content, reports and alerts.

We have prototyped and validated a framework for such an SMS and propose that an Open Source project could use this to implement a European solution to this global problem.

The Information Deficit

On a typical software development project, a manager establishes schedules and processes with the team members who use role-specific tools to evolve a code-base from which product increments are ultimately released.

The manager attempts to track progress by soliciting and merging data from the various roles (requirements, engineering, test, design …).

The other team members each have their own view of the code-base, specific to their own role and set of tools.

Since the primary source of management data is team members, it is second-hand, largely subjective and necessarily inaccurate.

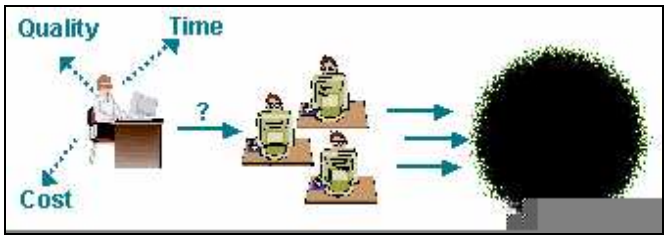


Figure 1 - Software - the invisible product

As time goes by, well-meaning engineers often try to provide good news by adjusting the project activities without moving the deadline. They probably had built in "fat" to the initial schedule for this very purpose.

Alternatively, engineers and team-leads may be liberal in their interpretation of task "completion". After all, a software component is sort-of complete when it is implemented, unit tested, integrated, user tested … And there is all kinds of latitude in the meaning of "tested".

Worst of all, when under pressure, programmers can use many shortcuts to finish a component on-time, but at the cost of quality. This can deceive a manager that his project is on schedule, whereas in reality he is accumulating a deficit that will cost much more later – during testing, integration, future iterations, maintenance, etc. This practice, if widespread, has caused many projects to fail disastrously, necessitating huge delays in product release, the rewrite of significant proportions of the application and/or the release of a very disappointing product.

Two projects may be reported to be 80% complete, but there could be significant actual differences in completeness and risk. This would be exposed by standardized data on essential indicators such as design compliance, code quality, test coverage, volatility, the rate of new code versus changing of existing code etc.

While the goals of developers and managers are well aligned, seem the basis of good division of responsibility, are mutually empowering, and provide the basis for good, self-aware and improving engineering, the lesson learned is that it takes more than good intentions to make a project succeed. It takes actionable information – and this, compared to the sea of non-actionable information, is generally unavailable.

A New Kind of Information

The required information does exist in the development environment. However it is buried in the code, the CM system, the file system, design tools, bug-tracking tools and a range of other activity-specific tools. There are tools, such as Maven, that collect reports from different

tools into a single location. This is valuable to the development team and Maven has achieved a reasonable adoption rate.

However, the information of most value is often drawn from multiple sources and trackable over time.

For example:

- The real indicator of integration risk is as measured not by what is checked-in and working on the mainline, but rather by how much is checked-out on developer branches, to what degree the branches overlap, and what proportion of the interfaces for the key abstractions in the system are changing.
- Knowing the number of violations of complexity thresholds is less interesting than whether the total number is increasing or decreasing, or which developers are contributing most to the increase or decrease.
- I may know to which projects my principal engineers are assigned. I also know that their time is heavily taxed helping to put out fires on other projects. Are they spending their valuable time on areas that are important to the business, on projects that are using new technologies, coaching less experienced developers? Or are they constantly dragged onto tasks just because they are urgent, tasks that do not really use their experience?
- 100% test coverage is not practical, but 30% coverage that includes the most complex and most changing parts of the code is better than 60% that only test the unchanged code.
- What is the bug density, not just simplistically, but by subsystem, or by customer category; are bugs in certain subsystems taking longer to fix, or impacting more files; is the test suite efficient; is the resource split between servicing defects and developing new features acceptable; where is the mounting code needing refactoring?

This paper is not another proposal for a software development process, forward-engineering/planning tool, or reverse-engineering tool. Instead, it describes a revolutionary metaphor for actionable information – the software map. The Software Mapping Server (SMS) delivers information from the map to managers and developers, at different levels of detail to support their unique needs.

The SMS supports agile development processes and traditional ones, and recognizes there is no need for additional forward-planning, nor is there a need to reverse engineer the code, because the code, itself, has the answers.

We start with bottom-up information from various systems, abstract it to the appropriate level of detail, correlate it to the truth – the code – and present it in either an alert, dashboard, or browsing medium.

Actionable Project Information

The key attributes of information that make it actionable are listed below.

- **Reality-based**. Information that is derived from aspirational artefacts such as project plans, designs or test plans are of limited use unless correlated with hard data from the development environment.
- **Aggregated**. We need to integrate data from multiple domains, such as source, version control, test, etc., in order to make the information truly actionable.
- **Analyzed**. A lot of time can be spent wading though vast amounts of data from diverse sources in order to discover the answer to specific questions. The data needs to be analyzed, prioritized and filtered so that the actionable information it contains is quickly recognized and used.
- **Relevant.** Actionable information is in the eye of the beholder, requiring flexible definition, collection, and presentation.
- **Accessible.** Actionable information sometimes occurs as events (alarms), and sometimes results from analysis (browsing), so both need to be supported in a way that respects the user's desired level of detail. Also, the user should not need to be familiar with any more than standard office-automation tools to access the information.
- **Time based**. It is usually more effective to scan information about what has changed rather than reviewing all the information about the current state of a project to spot actionable information. The SMS must continuously or regularly collect data for analysis and presentation.
- **Automatically extracted**. Manually assembled analyzed, analyzed and presented information is expensive and time-consuming to generate. Often it is not obvious whether the information is relevant until after it has been generated. Manual generation is simply not sustainable as part of an ongoing process.

SMS Architecture

The architecture of is open, XML-based and easily extended.

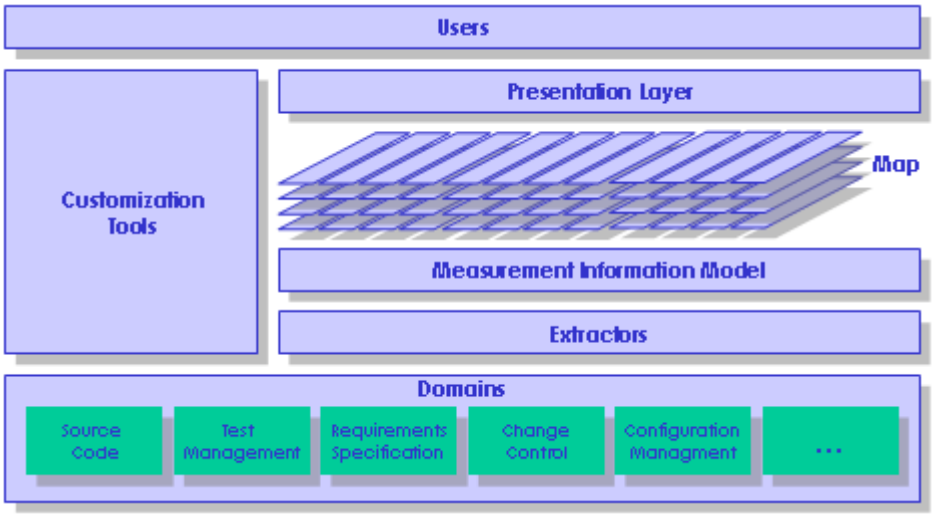


Figure 2 - Software Mapping Server (SMS) Architecture

- ***Domains*** are not a part of the SMS as such. They are the source code, development environment and project management tools in use on the projects at a customer site. They are the source of much of the information presented to the users by the system.
- ***Extractors*** extract information from a specific tool in a specific domain and add it into the Map (through the Information Framework). The need to implement an extractor for each of the many domains that exist in the software development industry points to the Opensource development model.
- The ***Measurement Information Model*** defines how data from the various domains is combined into higher-level information. For example it could define a "quality" metric for each source code item based on the elemental metrics generated by source code extractors, or define a "Productivity" metric for each employee based on the "Quantity" and "Quality" of source code he or his team

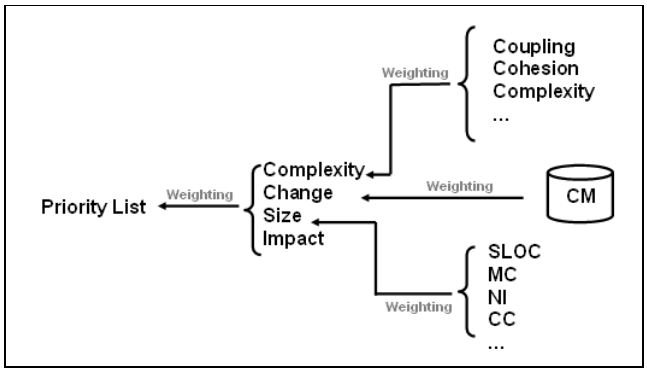


Figure 3 - Example Measurement Information Model

generates, as gleaned from the version control extractors. This is fully customizable.

- The ***Map*** is at the centre of the SMS. As in Geographical Information Systems (GIS), the Map is built out of layers. Where GIS layers may be topology, roads, electricity grid, water grid, cover type, etc., SMS layers correspond to source code, employees, work, etc. Unlike a GIS system that shows the state of an area at a single point in time, however, the software Map also tracks how the software project is progressing over time. This is performed by recording Snapshots at discrete intervals (e.g. weekly).
- The ***Presentation Layer*** defines how information contained in the Map is presented to the user via a Map Browser, Reports, Dashboards and Alerts.

Prototype Results

The SMS architecture has been largely validated by several man-years of prototyping effort. This implements the Map in a relational database. The Presentation Layer is largely XSL-based. Interfaces predominantly use XML.

Figure 4 shows the type of output achieved using a browser interfact to the SMS. This allows the user to understand how key metrics are changing over time, how they have changed since a specific point in time, and how values are distributed across projects and components.

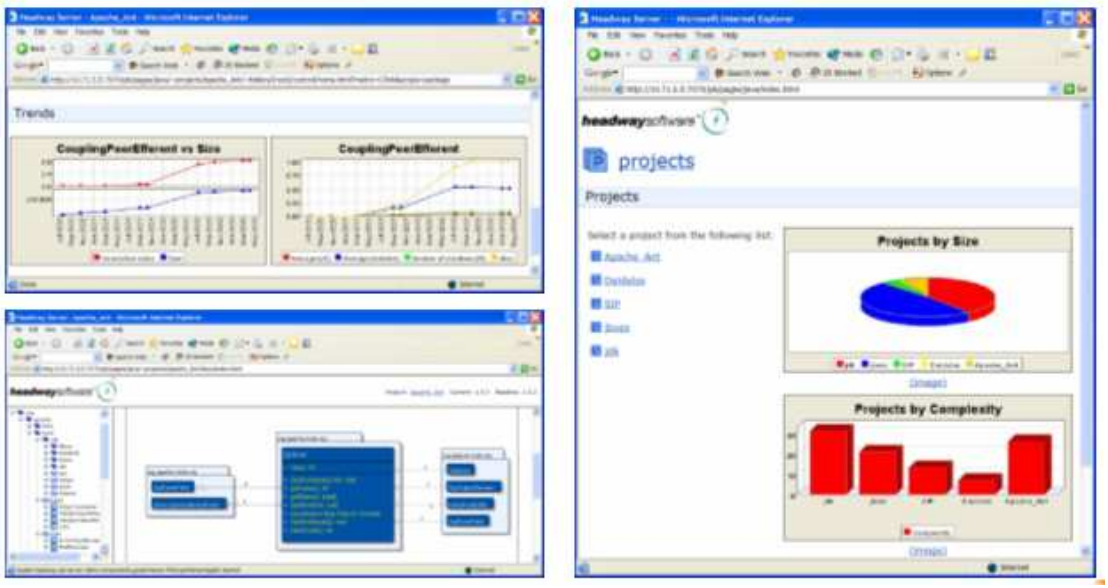


Figure 4 - Browser Interface to SMS

Customizable templates define the content of reports. These can be specified on a role/event (e.g. Project Manager/ release review meeting) basis, and generate manager-friendly formats such as html, pdf and Word.

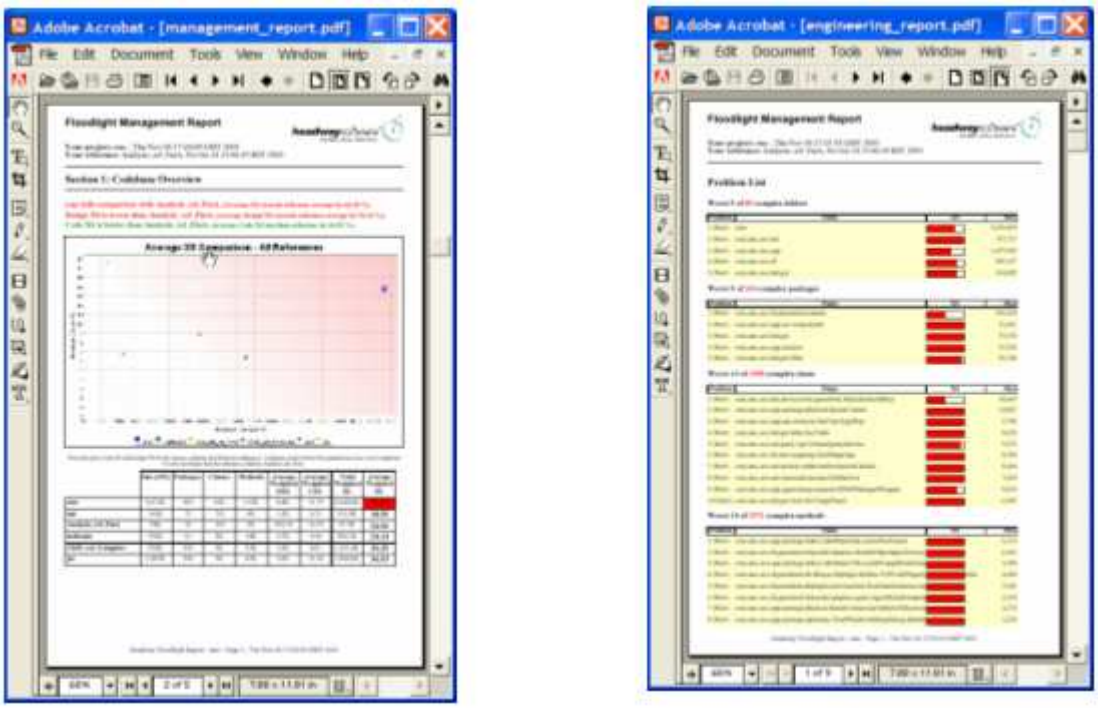


Figure 5 - PDF format reports

Conclusion

There is a dearth of actionable information available for decision-makers on software projects. The provision of such information promises to have a hugely positive impact on the productivity of development teams, and yet adequate solutions have not yet surfaced.

We propose a framework that is scaleable and that has been validated through the implementation of a prototype server. The main challenge to implementing a full SMS that can work for heterogeneous development environments is the creation and maintenance of a wide range of data extractors.

We believe that the Opensource development model is ideally suited to this problem, and that an IST-initiated OSS project could result in a European solution that is a major improvement on current alternatives.

References

[Bansiya02] Bansiya, J., Davis, C.G., *A Hierarchical Model for Object-Oriented Design Quality Assessment*, IEEE Transactions on Software Engineering, January 2002, pp. 4-17.

[Beck99] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison Wesley, 1999.

[Chidamber94] Chidamber, S.R. and Kemerer, C.F., *A Metrics Suite for Object Oriented Design*, IEEE Transactions on Software Engineering, June, 1994, pp. 476-492.

[Chidamber98] Chidamber, S.R., Darcy, D.P., and Kemerer, C.F., *Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis*, IEEE Transactions on Software Engineering, Aug 1998, pp. 629-639.

[Fowler99] Fowler, M., *Refactoring*, Addison Wesley, 1999.

[Fowler02] Fowler, M., *Reducing Coupling*, IEEE Software August 2001, pp. 102-105.

[Hunt99] Hunt, Andrew, and Thomas, David, *The Pragmatic Programmer*, Addison-Wesley, Oct 1999, ISBN: 020161622X

[Lieberherr89] Lieberherr, K.J., Holland, I.M., *Assuring Good Style for Object-Oriented Programs*, IEEE Software, September 1989, pp.38-48.

[Martin02] Martin, Robert C., *Agile Software Development, Principles, Patterns, and Practices*, Prentice Hall, 2002.

[McCabe94] McCabe, Thomas J. & Watson, Arthur H. *Software Complexity*, Crosstalk, Journal of Defense Software Engineering 7, 12 (December 1994): 5-9.

[McGarry01] McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., Hall, F., *Practical Software Measurement – Objective Information for Decision Makers*, Addison-Wesley, 2001.

Perspectives for a Model-driven Service Engineering Discipline

Claus Pahl *
*Dublin City University, School of Computing, Dublin 9, Ireland

Abstract. *The notion of services has become ubiquitous in recent approaches to software engineering. Services and processes are abstractions that suit the modelling requirements at different stages of the software systems development process. We propose a model-driven service engineering approach based on the model-driven architecture framework and adapted to the specific needs of the services context and service-oriented architecture as the principle of the deployment platform. Service process composition and semantic modelling are here central aspects. We present solutions for a services engineering framework and outline perspectives that need to be addressed in a services engineering discipline.*

Keywords: Service-oriented Architecture, Process Composition, Model-Driven Architecture, Service Ontology, Web Services

1 Introduction

The service notion has become omnipresent in recent discussions about software platforms and software engineering approaches. Services as a deployment paradigm is at the core of service-oriented architecture (SOA). The Web Services framework (WSF) is the central platform implementation for this paradigm [ACKM04]. SOA deals with distribution, interoperability, and process description and deployment. The service concept is also used to capture business and workflow processes at a higher level of abstraction, developing services based on flow-based processes. A business service is the realisation of a business goal. Although both have their specific requirements and implications, both aspects raise the issue of an integrating discipline of service engineering.

Model-driven architecture (MDA), promoted by industry bodies such as the OMG, proposes an approach that, although not specific to the services context, can provide a framework for this discipline [Obj03]. MDA focuses on maintainability through model-centricity and on automation of programming activities through code generation. MDA can be adapted to the services context by, firstly, focussing on the WSF as the platform, secondly, enhancing the semantic modelling capabilities, and, thirdly, improving the service reuse aspect. Although ontologies [DOK03] have predominantly been used to support data-intensive applications, we make the case here for their use as a modelling technique in a service- and process-oriented context [Dju04, GJH05, Pah05a]. Based on our experience resulting from several

case studies, we discuss a service engineering discipline along the following perspectives: service development and deployment, rigour and formality, and methods and techniques. We look at each of the perspectives addressing the aims, challenges, and opportunities based on the case studies we have been involved in. Our objective is to identify cornerstones of a service engineering methodology using the discussion of the individual perspectives. We refer to the case studies to illustrate our experience and how it has influenced the design of our approach.

2 Platform and Software Engineering Implications

Service-oriented architecture as the architectural platform and the Web Services framework as the deployment platform have implications on a corresponding service engineering framework. A *service* is a software component provided at a given location. Services are usually used 'as-is', based on abstract description published by the provider in directories and used by potential clients to locate suitable services. Several aspects characterise the targeted service platform [ACKM04]:

- Distribution: This deployment aspect characterises the services platform as a distributed infrastructure. Handling invocations is its central task.
- Independent deployment: This development aspect refers to the independent, blackbox deployment of services where different organisations are involved as clients and providers. This requires suitable description techniques to communicate service requirements and properties. Additionally, trust between the actors is an issue in these open, unconstrained environments.
- Process-orientation: The development of services is tightly linked to the notions of architecture and process. The composition at various levels ranging from business workflows to service processes is central [PV02].

Cost effectiveness and cost reduction are the primary drivers of current software technology development. Reuse is a method to achieve reduced costs and sustain or improve quality. Automation is a method to reduce time-to-market and to improve maintainability. MDA emphases automation and encourages model reuse. SOA focuses on reuse-as-is in service form.

We propose here cornerstones of a *discipline of model-driven service engineering*, characterised by two specific modelling foci. Firstly, composition-centric modelling shall address services, processes, and layered reuse. Secondly, ontology-based semantic modelling shall address formality, exchange and collaboration, and automation. This is influenced by various industry-led initiatives such as MDA (OMG), Ontology-driven Architecture ODA (W3C), and the Business Process Modeling Notation BPMN (OMG).

We have identified a number of perspectives by reviewing different case studies, which are facets that characterise the discipline and that reflect aspects that have impacted the design of our proposed development approach.

- Rigour and Formality – the foundations of the modelling notation and techniques in terms of ontology technology.

- Service Development and Deployment – the software process lifecycle with its stages and activities based on Model-driven Architecture.
- Methods and Techniques – composition-centric modelling and service architecture to support the activities.

Three *project case studies* have influenced the design of the proposed methodology and our discussion of a path towards a discipline of service engineering.

- An e-learning system consisting of platform and content services is being re-engineered to a services platform. The project focuses on an in-house developed system and is motivated by maintenance problems that arose after several years in service while undergoing constant changes and extensions.
- An application service provider (ASP) infrastructure was re-engineered to a portal-based service platform. A large number of software providers provide their services now as Web services allowing different clients to access services through a customisable service portal. This empirical case study is a typical example of legacy systems integration. It provides challenges in terms of data modelling and data integration in heterogeneous service-based environments.
- A services-based online banking application has been used as a conceptual case study to address architecture topologies and patterns. Service and process compositions on an abstract, logical level need to be mapped onto possible distributed architectures with different interaction mechanisms.

3 Service Development and Deployment Process

The first of the perspectives is software process-centric. The overall development and deployment process is based on general constraints of the services context and our experience, see Fig. 1. The banking case study has highlighted the importance of service engineering in terms of business processes and workflows. Standard procedures of account management have to be implemented. The e-learning case study has drawn attention to the deployment and code generation aspects. Current reusable learning objects and services are widely discussed, raising the importance of publishable semantic descriptions.

- Business service modelling. The recent discussions of MDA have increasingly focused on this layer that addresses business objects and processes at a high level of abstraction.
- Architecture modelling. The visual definition of service, processes, and their architecture needs to be based on a UML notation in order adopt to current practice and to allow the reuse and integration of existing models.
- Service process analysis. Process composition based on services from different sources requires a rigourous analysis of structural and behavioural properties.
- Process composition. The implementation of abstract process definitions in a services-based deployment context requires again rigourous matching analysis between required and provided services.

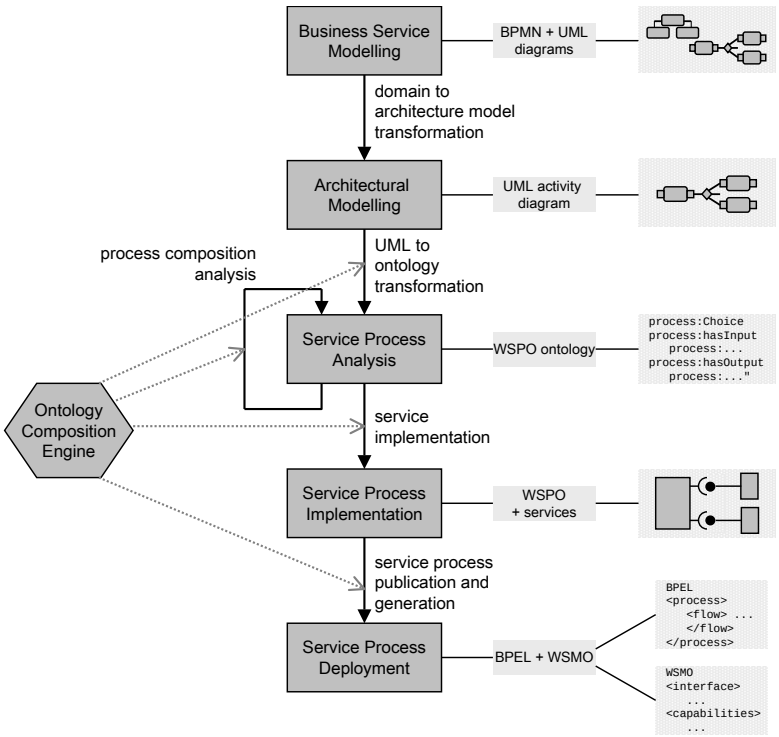


Figure 1: Development and Deployment Stages with corresponding Techniques.

- Service process deployment. Automated code generation for services comprises of executable process specifications and abstract service descriptions.

A layered modelling approach addresses different abstraction levels. Process composition is a central activity at different stages. Code generation creates executable and non-executable code. Tool support in form of a composition engine is required for the modelling, composition, and code generation activities.

4 Rigour and Formality

The second perspective looks at foundations. Rigour is a central requirement for two reasons. Firstly, extensive modelling is required in an environment based on collaboration and exchange of information. Precision and detail are here mandatory. Secondly, the automation of analyses and code generation at development time and deployment activities at runtime equally requires formal techniques.

- Service and process modelling is traditionally well supported by formal techniques. Process behaviour is supported by process algebras and calculi. The composition of services is supported by component matching and interface refinement techniques. A wide range of analysis and reasoning techniques (including deadlock and matching analysis) are available.

- Formal semantics for a modelling notation also supports other aspects such as the exchange and publication of models in a common format and the (horizontal and vertical) transformation of models.

Semantic service description, composition, and matching is the aim. In particular, the ASP project required the semantic integration of information. Data from various legacy systems had to be integrated in order to facilitate the flexible and automated composition of services to suit the needs of individual clients.

We propose an ontology-based formal foundation as the solution for these requirements. We argue that this foundation suffices to integrate the different aspects just outlined and that it addresses the aspects adequately. Description logic, which gives a foundation for ontology languages such as OWL-DL, is the proposed formal language [BMNS03]. It provides interoperability and enables a multilayered framework through ontologies. It supports process description and composition through extensions of classical description logics [Pah05a]. This formal foundation is based on the following principles of ontology-based knowledge representation and its extension for service and process modelling and composition:

- Concepts representing entities of a domain and relationships between these concepts that explain the properties of concepts are the cornerstones of ontology-based modelling.
- An extended relationship subexpression language using process combinators realises process expressions that characterise accessibility relations between states of a system, where the latter ones are the concepts of the model,
- Additional extensions can cover data aspects by introducing names to represent e.g. parameters.

Reasoning is based on a subsumption (subclass) relationship at the core. Necessary refinement and simulation notions for service matching and process analysis can be mapped into subsumption-based reasoning. For instance, a simple dynamic logic – a modal logic of programs – can be represented in description logic that addresses service matching [Pah05a].

The complexity of composition in an architecture-centric setting is a major challenge [ACKM04]. Service ontologies such as OWL-S and WSMO [PL04] are good starting points; the research into the Web Service Process Ontology (WSPO) which focuses on service composition has, however, shown some limitations [Pah05b]. Still, we consider ontologies and their underlying logic sufficient to deal with the required composition analysis and matching tasks, although sometimes not as powerful as other dedicated formal frameworks such as process algebras.

5 Methods and Techniques

The third perspective looks at the methods and techniques of an engineering discipline. The method has service modelling at its core. The aims are to achieve cost reduction by providing a framework for reuse based on service-orientation as the concept, and by providing a framework for maintenance based on layered, automated transformation. Ontologies allow us to integrate both strands.

The techniques are based on an ontology-based composition engine. This will provide description, reasoning, and transformation techniques. A number of constraints have to be considered:

- UML is an accepted IT sector standard for modelling.
- Service-oriented architecture with the Web Services platform is the trend in enterprise application integration (EAI) and legacy systems integration.
- Ontologies are a standard for semantic domain and services modelling.
- Formal analysis and reasoning for composition has been extensively researched.
- Automated transformation and code generation is at the core of MDA.

The service platform is not about implementation, which shifts the focus of services engineering to an architectural level. The abstract properties of services and their architectural composition to processes in terms of platform features is central. Topologies and patterns of process interaction are key aspects.

Legacy systems integration in the ASP-project required process composition support for individual services. The need for semantic descriptions at domain- and also service-level arose in the e-learning case study, where discovery and matching of provided services usually starts with domain-specific aspects, before the services are looked at in terms of their functionality.

The role of models for service engineering is central and crucial. In addition to the development perspective, models are the basis of published service descriptions and also contracts between providers and clients of services. We will focus on the architecture and platform modelling layers now.

5.1 Visual Modelling

Based on the MDA philosophy, modelling needs to be supported at different layers, here specific to the services context. Business service modelling (called computation-independent in MDA) is the first, followed by architecture modelling (platform-independent in MDA), and finally process execution and service description (platform-specific in MDA).

The objective at the architecture layer is the definition of service architectures as processes. Extensions of UML activity diagrams customise the diagrams for the semantic modelling of service processes [GJH05, Pah05b]. This would add semantic annotations such as preconditions and postconditions (effects) to individual services. OCL provides a basic framework [WK03], but service ontologies enable richer and more comprehensive description and reasoning [PL04].

5.2 Formal Reasoning

Composition is an activity that occurs in two forms in the services context. Firstly, process analysis: the abstract composition of individual services to processes requires some constraints to be satisfied, e.g. pipe-based combinators require the output of the first element to match the input of the second. Sometimes, postconditions of one process element have to satisfy the precondition of the next element. Semantic annotations provide the necessary information; a refinement-based reasoning approach provides the matching support.

Secondly, process implementation: each service element of an abstract process definition has to be implemented by a concrete service that matches its syntactical and semantical requirements. Various

properties are used to determine the closest match. Based on semantic information, a refinement-based inference system (e.g. weaken the precondition, strengthen the postcondition) as it is realised in services ontologies such as OWL-S or WSPO can support this activity.

The proposed ontology-based composition engine, see Fig. 1, can implement these two reasoning activities. Semantic Web and ontology technology creates an opportunity as continuing research and support in this area can be expected due to the commitment of industry and standardisation bodies.

5.3 Transformation

Transformations occur in two contexts. Firstly, horizontal transformation refers to the transformation of UML models into ontology representations (or vice versa). UML activity diagrams consist of activity nodes and edges using control flow nodes to describe control and data flow. These models can be mapped to a WSPO ontology with its service elements and process combinators. UML models and their extensions can be semantically defined and analysed using ontologies.

Secondly, transformations between the stages including the final code generation step are the forms of vertical transformation. The last step is platform-specific, which in the SOA case means executable service process definitions and publishable service descriptions. The latter ones are ideally service ontology-based, which makes an ontology-based framework suitable. Executable service process definitions in WS-BPEL can almost directly be generated from the ontology-based process representations.

6 Conclusions

We have introduced service engineering as a multi-layered, multi-stage activity, ranging from defining business services based on workflow processes to platform-specific service implementations. In addition to standard software industry aims of cost reduction through automation and improved maintenance, service engineering requires semantic integration and process-orientation as central elements, both relying on composition and transformation techniques.

- Semantic service and process models can form the missing link between domain-level matching (for example in e-learning where learning objects can be searched in repositories based domain-level annotations) and syntactical service interface matching (for example in ASP where service repositories can be searched based on WSDL descriptions).
- Processes based on services are essential abstractions across all stages. MDA supports this layered development from semantic modelling to service deployment; specific business process modelling and execution languages are available in form of BPMN and WS-BPEL at the different layers.

Ontologies play a central role in integrating these aspects. Their capability as an abstract modelling tool is obvious, which, as we hope to have motivated, can extend to computational concepts such as services and processes. The formal, logic-based foundation makes ontologies also a suitable tool for defining and reasoning about composition. These observations, that have led us to the organisation of a services engineering discipline, are based on empirical research that we have carried out. While an

ontology-based modelling and composition can address central problems, a discipline needs to reach further. Interoperability, cooperation, and trust are perspectives that characterise this wider context.

A services-centred discipline also needs to shift the traditional focus of software engineering approaches. A software value chain can be distinguished into stages which follow a top-down abstraction hierarchy. While in the past the main attention has been put on lower program and software-related aspects, elements higher up also need to be addressed. This includes workflow and business process modelling and overall semantic integration. The current focus on business modelling and analysis and corresponding notations – such as BPMN – is one of the activities demonstrating the importance of this aspect.

References

[ACKM04] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[BMNS03] F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[Dju04] D. Djurić. MDA-based Ontology Infrastructure. *Computer Science and Information Systems (ComSIS)*, 1(1):91–116, 2004.

[DOK03] M.C. Daconta, L.J. Obrst, and K.T. Klein. *The Semantic Web*. Wiley, 2003.

[GJH05] R. Grønmo, M.C. Jaeger, and H. Hoff. Transformations between UML and OWL-S. In A. Hartman and D. Kreische, editors, *Proc. Model-Driven Architecture – Foundations and Applications*, pages 269–283. Springer-Verlag, LNCS 3748, 2005.

[Obj03] Object Management Group. *MDA Model-Driven Architecture Guide*. OMG, 2003.

[Pah05a] C. Pahl. An Ontology for Software Component Matching. *International Journal on Software Tools for Technology Transfer (STTT), Special Edition on Component-based Systems Engineering.*, 7, 2005. (in press).

[Pah05b] C. Pahl. Ontology Transformation and Reasoning for Model-Driven Architecture. In M. Kifer and S. Spaccapietra, editors, *Proc. Intl. Conference on Ontologies, Databases and Applications of Semantics ODBASE 2005*. Springer-Verlag, 2005.

[PL04] T. Payne and O. Lassila. Semantic Web Services. *IEEE Intelligent Systems*, 19(4), 2004.

[PV02] F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *ACM Transactions on Software Engineering*, 28(11):1056–1075, 2002.

[WK03] J.B. Warmer and A.G. Kleppe. *The Object Constraint Language – Precise Modeling With UML*. Addison-Wesley, 2003. (2nd Edition).

Emergent Phenomena in AmI Spaces

Ioannis D. Zaharakis[1] and Achilles D. Kameas[1,2]
[1] Computer Technology Institute, Patras, Hellas
[2] Hellenic Open University, Patras, Hellas

Abstract. *This work aims at a) identifying the forthcoming changes in our everyday life due to the ever-increasing level of complexity that inculcates our interactions with the devices surrounding us, b) introducing a bio-inspired world model (framework) that deals with different perspectives of the interrelations developed in symbiotic ecologies where people and artefacts coexist, and c) proposing a high level architectural scheme of an AmI space reflecting the basic ingredients of the future indoors/outdoors applications based on Swarm Intelligence and Complexity Science.*

Keywords: Complex Systems, emergent behaviour, Ambient Intelligence, Swarm Intelligence, Ubiquitous Computing

1 Introduction

The vision of Ambient Intelligence (AmI) implies a seamless environment of computing, advanced networking technology and specific interface ([9], [11]). In one of its possible implementations, technology becomes embedded in everyday objects such as furniture, clothes, vehicles, roads and smart materials, and people are provided with the tools and the processes that are necessary in order to achieve relaxing interactions with this environment. The AmI environment can be considered to host several Ubiquitous Computing (UbiComp) applications, which make use of the infrastructure provided by the environment and the services provided by the AmI objects therein.

An important characteristic of AmI environments is the merging of physical and digital space (i.e. tangible objects and physical environments are acquiring a digital representation). As the computer disappears in the environments surrounding our activities, the objects therein become augmented with Information and Communication Technology (ICT) components (i.e. sensors, actuators, processor, memory, wireless communication modules) and can receive, store, process and transmit information [11]. The AmI objects differ from traditional objects in that they can communicate with other AmI objects and can interact with the environment. Of special interest is the information that AmI objects process, which can be descriptions of the context of use, data to be used for a task, guidelines on how to perform a task, messages to be sent or that have been received from other objects. The result of information processing is

a set of services, that is, a set of abilities that appear in the digital space and relate to information.

On the road to becoming an AmI space, our living space as of today is already populated by many devices, which are able to process (and sometimes store) and communicate digital information. These are divided in three major categories [8]:

- The PC Internet world where PC and PC peripherals communicate.
- The broadcast world that serves set-top boxes and traditional consumer electronics, e.g. TVs, VCRs, stereo systems, CD/DVD players etc.
- The mobile world, consisting of multimedia mobile phones, PDAs, laptop computers and similar devices, that provides unparalleled connectivity and freedom of movement into and out of the home environment.

A new fourth category, as a consequence of the gradual realization of the AmI vision, is the white-goods devices, like refrigerators, microwaves ovens, air-conditioners, etc, which recently is including everyday objects enhanced with sensing, processing and communication abilities.

This work builds upon the envisaged structure of AmI environment as one populated by thousands of communicating tangible objects and virtual entities [13]. Following an agent-oriented programming approach [21], we can classify them into active, in the sense that they have an explicit goal to achieve (i.e. a TV has a goal to display a selected broadcast, an air-conditioner has a goal to maintain a certain air temperature, etc) and passive. The latter are those being used as part of tasks (i.e. numerous everyday objects surround us like tables, chairs, walls, photo frames etc), according to the plan of active objects (e.g. a wall may be used to hang a poster out or to project a video). At a minimum, AmI environment will contain network infrastructure will be available that will make anytime, anyplace (within boundaries of acceptability) interaction among and with these objects feasible.

2 Research Issues and Requirements

The heterogeneity of AmI objects make necessary the development of middleware systems on top of which UbiComp applications can function transparently with respect to the infrastructure [13]. In order to preserve the autonomy of AmI objects and to cater for the dynamic nature of UbiComp applications, ad-hoc networking has to be supported; thus, no specific network infrastructure can be taken for granted. The underlying physical networks used are heterogeneous ranging from infrared communication over radio links to wired connections. Since every node serves both as a client and as a server (devices can either provide or request services at the same time), the required communication can be considered as peer-to-peer [12].

As a consequence of the dynamic nature of UbiComp applications and of the mobility of AmI objects, the middleware has to use services and capabilities with changing availability

[14]. In addition, even a service that is both functional and reachable can become unavailable (the volatility problem). As objects of all sizes are candidate components of ubiquitous computing applications, the enabling middleware has to be adaptable to the physical properties (e.g. size, power) in the case of tangible objects and computational abilities (e.g. memory) of a broad range of devices [10].

Since UbiComp applications operate within an extremely dynamic and heterogeneous environment, the context definition, representation, management and use become important factors that affect their composition and operation. UbiComp applications have to dynamically adapt to changes in their environment as a result of users' or other actors' activities. To ease the development of such applications it is necessary to decouple application composition from context acquisition and representation, and at the same time provide universal models and mechanisms to manage context [7].

Current digital paradigms for interacting with the digital world are often inadequate and are leading towards a dead-end of frustration-in-use and impoverished digital living. This situation will deteriorate once computers become dispersed into everyday environments; interaction problems will be significantly multiplied and shift towards a different, under-explored territory of paradigms that originate from having to interact with hundreds interacting nodes, many of which will be physically imperceptible. Additionally, as computational power diffuses in our living/working environment and the everyday devices that are capable of sensing, processing and communicating continuously grow in numbers, new requirements are posed by i) the people who use UbiComp applications, ii) the heterogeneity of the involved devices, and iii) the large number of the involved devices.

As a consequence, key research challenges have to focus in services availability including both services aimed at end users as well as machine to machine services, and to deal with dynamic composability and adaptability, context awareness, autonomy and semantic interoperability. Essentially, new research issues arise concerning i) the system complexity emerging by the thousands local interactions between people and artefacts, ii) the need for flexible and dynamic system architecture capable to evolve and adapt to new situations and configurations, iii) the context dependence of the exchanged information, and iv) the human involvement and especially new, more natural, human-machine interaction schemes.

The abovementioned features impel to the development of a framework that will help ordinary people deal with the complexity of using UbiComp applications that will exist within AmI environment, and especially assist them in dealing with the interactions occurring therein. The framework must be capable to host and reflect transparently the available services as well as the potential use of the participating objects. Although the available services may somehow be exhibited, the potential use of the objects emerges mainly from the interactions of the humans with the active and passive devices and these interactions are not only time-dependent but also space- or context-dependent. Specifically, the framework should employ a scheme based on five premises:

- An easy-to-understand-and-use end-user programming model, which will build upon known world models, interaction metaphors and usage concepts.
- The exploitation of locally stored knowledge into the surrounding devices as well as the AmI environment.
- A classification mechanism based on attributes of social behaviour of the components of UbiComp applications.
- Procedures for distributed decision making, which will facilitate the emergence of UbiComp applications as compositions of collaborating services, with or without the explicit intervention of the humans.
- A composition mechanism of previous successful/failed actions aiming at feeding back the UbiComp applications.

The next sections set the scene of a near future everyday living/working environment and describe an engineering approach inspired by biological structures capable to deal with phenomena arising in such an environment. Subsequently, a related high level architectural scheme is introduced. A depiction of the involved technologies and associated research directions follows and the paper concludes summarising the most prominent elements of this work.

3 Proposed Conceptual Framework

A living/working AmI space comprising of many heterogeneous objects with different capabilities and provided services could be considered that is populated by a *heterogeneous swarm*. There are many potential benefits of such an approach including greater flexibility and adaptability of the system to the environment, robustness to failures, etc. The swarm will comprise different typologies of societies, and so it will be heterogeneous also from the provided services point of view. Such differences will contribute to the overall capabilities of the system. As a general principle, the services should be as simple as possible, according to the concept of summing the capabilities of extremely simple members by the swarm system increasing the number of agents, sharing the resources and maximizing the effectiveness of the communication. Ideally, the composition should emerge based on previous interactions and on the context (time and place) they took place.

From a macroscopic perspective, a natural system (or ecology) consisting of thousand living organisms exhibits superiority, in terms of stability, coherency, flexibility and adaptability, because these organisms are integrated and optimized with respect to their computation and control strategies, morphology, materials, and their environment (see details in http://www.neuro-it.net). The participants (or organisms) are deployed in such an ecosystem where coherent choices are manifest across the whole space of options, rather than just at the computational/control level. These organisms live, obey rules and reap the benefits of a society of kin. Societies may vary in size and complexity but they share a common

property: they provide and maintain a shared culture. Intelligent creatures create and refine social rules in order to perpetuate the society. These rules constitute a culture which is communicated and shared by the society, and has important effects on the individual members. In this context individual intelligence needs to be analyzed within its social and therefore cultural environment.

An ecology is defined by the environment it resides in, the members it consists of, and the interactions between the members and the environment. In detail, an *ecology* is i) concerned with the interrelationship of organisms and their environments, ii) the totality or pattern of relations between organisms and their environment. Thus, the members of an ecology are the *organisms*, meaning i) complex structures of interdependent and subordinate elements whose relations and properties are largely determined by their function in the whole, ii) individuals constituted to carry on several activities by means of organs separate in function but mutually dependent. Advancing in more detailed decomposition, each organism is composed of *organs* that are i) differentiated structures performing some specific function in an organism, ii) bodily parts performing a function or cooperating in an activity, iii) parts of an organism that have been adapted to perform a specific function. Finally, the fundamental ingredients of the organ are the *cells*, which are elementary units capable alone or interacting with other cells of performing specific functions, and forming the smallest structural unit of a matter capable of functioning independently.

For engineering such problems and in an attempt to create a metaphor of the biological structures and principles into the information systems, the traditional Artificial Intelligence (AI) focused on addressing intelligence as an individual phenomenon. This approach considers (intelligent) agents with cognitive states which maintain a (partial) model of the world they inhabit in and a (partial) model of the others. These agents are usually autonomous, social and try to accomplish tasks they are designed for [23]; in any case, the deliberation and the activation of these kinds of agents are based on their maintained models of the world and of the others. Despite of many interesting results, the abstractions made by this approach, led to isolated and disintegrated solutions regarding the development of large-scale intelligent artificial systems. A radical different approach is based on the belief that intelligent behaviour is inextricably tied to its cultural context and cannot be understood in isolation. Indeed, many natural systems can be described in terms of many individually “simple” components, interacting in “simple” ways and influencing their neighbours, and yet, are able to exhibit “complex” overall system level behaviour; those systems that exhibit this “emergent” globally complex behaviour from simple components are referred to as “complex systems” [6]. In contrast to traditional Artificial Intelligence, Swarm Intelligence (SI) is defined as the emergent collective intelligence of groups of simple agents, or in more detail, SI is the property of a system whereby the collective behaviours of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge ([3], [15]). As an engineering approach, SI offers an alternative way of designing intelligent

systems, in which autonomy, emergence, and distributed functioning replace control, pre-programming, and centralization.

Thus, when focusing on situated social systems in dynamic and non-deterministic environments, it is very hard (if not aimless) to embody into each organism complete models of the environment and of the others. Alternatively, none explicitly represented world models could be considered ([4], [5]); all the necessary information is out there changing dynamically and as so the world is the model itself. All we need is the means to capture, qualify and exploit the information that surround us. In order to deal with the collective behaviour of large ecologies in situated domains, a recent approach is the analysis and synthesis of small pieces of primitive behaviours that result from individual interactions.

Inspired by the biological social systems (ecologies), the analysis of artificial swarm systems could range in different levels depending on the desired granularity. For example, in a macro-scale domain where many autonomous and heterogeneous agents interact, every single agent could be considered as a single behaviour building block which models a set of primitive behaviours [16]. Every single agent has specific capabilities and therefore the members of the society could play the role of sensors, actuators and computation building blocks leading to both physical and functional construction. In this two-level granularity, the agents are the organisms that constitute the ecology. In a three-level granularity, the specialised agents could form structures resembling to organs (e.g. for sensing, acting etc) and thus the swarm becomes an abstract loosely coupled organism and several swarms constitute an ecology. From another perspective, the society could be formed (or modelled) as a neural network (sensors/actuators building blocks representing the input/output neurons and computation building blocks representing hidden neurons) that can learn and evolve [17]. Now, the whole network is the ecology, the neurons are the organisms and the synapses between neurons represent the local interactions of the organisms. A third perspective is to consider the large amount of (different) artefacts as sensor networks owning limited power, computational capacities and memory. Sensor networks are densely deployed, have not global identification (ID), and their topology changes very frequently. Based on their (limited) processing abilities, instead of sending raw data, they may locally carry out simple computations and transmit only the required and partially processed data [1]. Hence, the combination of sensor networks with artefacts with computing and effecting capabilities may trigger the continuous formation of new societies that provide services not existing initially in the individuals and exhibiting them in a consistent and fault-tolerant way.

Independently of the analysis level, the computational capabilities (and the intelligence) of the ecology are distributed over the central “nervous” system, the peripheral system, the materials of the ecology’s body and the physical phenomena created by the interaction of the ecology with its environment. Putting such entities into a UbiComp environment could lead to **extelligent ecologies**, where knowledge and tools are diffused in the environment [22], underlying thus the **corporal literacy** of the ecology, meaning the awareness of the

extelligence and the working knowledge of all senses. This will pave the way for the generation of theory and technology of **synthetic phenomenology** (of the resulting ecology) meaning the understanding of the own self and its relation with the surrounding world (Fig. 1).

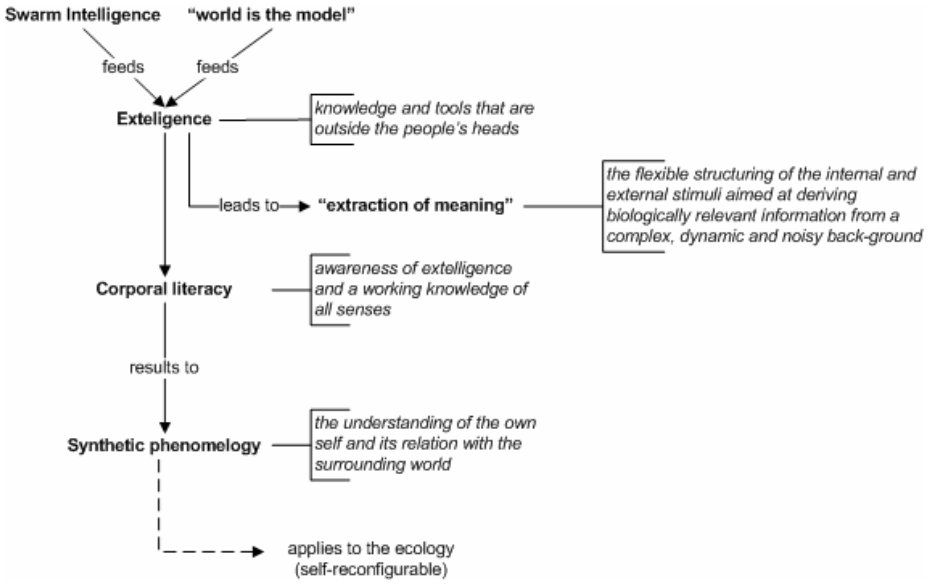


Fig. 1. Concepts relations on how dynamic changes of the environment and local interactions can lead to self-awareness.

Drawn from the above, the proposed high level architectural scheme that consistently reflects bio-inspired self-aware emergent symbiotic AmI space ecologies consists of the following ingredients (Fig. 2):

Basic building blocks: Everything can be regarded as a potential building block of a larger system, including sensors, hardware resources, software modules, artefacts. Every building block has an internal part, which is proprietary and possibly closed, and an external part, which is public or manifested as an influence to the environment, thus making the building block open to use or perceivable. Thus, building blocks are structures with physical and functional specification, capable to perceive the environment they reside in and to act upon it.

Ecologies: Groups of building blocks, their interrelationships and the associated environment form the ecologies. This means that ecology is more the configuration of the elements, rather than the elements per se. Although the members of the ecology have only local perceptions and local interactions, the ecology acts as a whole, which is not necessarily more than the sum of its parts but undoubtedly different from them (Gestalt Theory). The

behaviour of ecology is not determined by that of its individual elements, but where the part-processes are themselves determined by the intrinsic nature of the ecology.

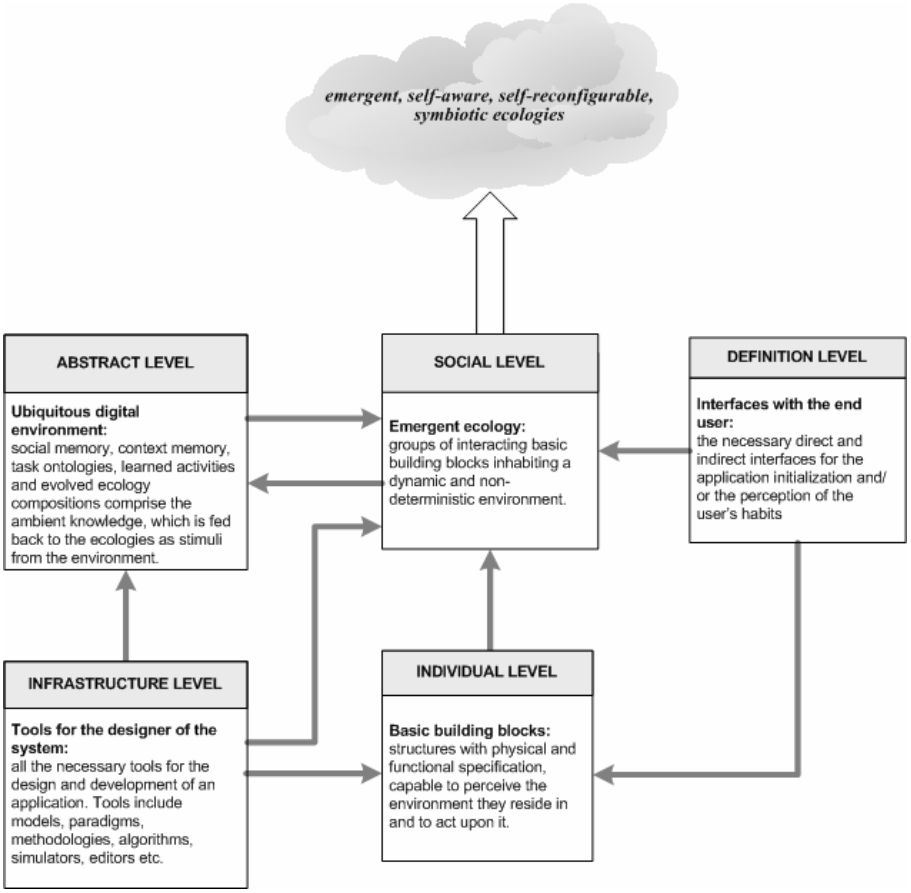


Fig. 2. High level architecture of the system. Individual and Social levels correspond to the basic building blocks and ecologies, respectively. Abstract level encloses the social memory of the ecologies; such knowledge must be transferred to the ecologies implicitly e.g. as stimuli of the environment, since individuals and consequently the emergent ecologies do not contain any knowledge representation scheme neither reasoning mechanism. Infrastructure level provides system designers with the appropriate tools to develop a system. Definition level is the user interface with the final user.

Ambient knowledge: It includes decentralised coordination models and selective interaction models which provide the abstractions and mechanisms for (i) environmental and context state reuse and reasoning, (ii) social and cultural memory representation, (iii) knowledge and experience interchange. Emergent behaviour, in this context, is considered to be as a result of some form of consensus on a shared view of the environment enabled by interactions among

heterogeneous, potentially arbitrary entities. Thus, ambient knowledge paves the way to emergent consensus as a substitute for social and cultural memory. Emergent consensus on a common view of a shared environment by interacting entities is a basis for establishing collective behaviour or complex adaptive system behaviour. Ambient knowledge is fed back to the ecologies as stimuli of the environment.

People: The involved users can be divided into several different categories such as building block developers, hardware designers, artefact manufacturers, applications developers and end-users. As each of these categories plays a different role in the system and has a variant perception for the world model, they are divided into two classes.

The first class includes the people that create the infrastructure of the system that includes models, theories, architectures, algorithms, behaviours, knowledge, protocols, mechanisms, interfaces etc. Thus, this category includes all the people that take part in the procedure of "creating" the building blocks and pose the driving force for the emergence of the ecologies. People of this category fall into architecture designers, building block designers and developers, knowledge engineers and hardware designers. Essentially, they perceive the world as a set of building blocks, architecture models, behaviours, shapes, proposed services, paradigms and guidelines, which will use in order to develop a variety of building blocks and ecologies.

The second class includes the end-users that use/cohabit with the ecologies. The end-users can play more than one role, as they can only use the provided building blocks and ecologies or they can co-create applications based on their own needs and desires. Thus the end-users can partly adapt and/or configure applications in order to compose personalised applications. The people of this category perceive the world as a set of symbiotic ecologies, which can be initialised and can learn and self-adapt according to their needs.

4 Instruments

Along these lines and aiming at building complete UbiComp systems, which make optimum use of distributed intelligence embedded in the periphery (sensors, actuators, body morphology and possibly materials), the involved theories, technologies and scientific communities are undoubtedly interdisciplinary. It is mentioned that the aimed outcome is the development of self-aware and self-reconfigurable symbiotic ecologies where artificial beings and humans coexist. The applications consist of tangible entities and ubiquitous services applied in indoors and outdoors areas. Particularly, the applications include autonomous software populating autonomous devices where the social interactions arise among the different elements and adaptation to unforeseen (at design time) situations encountered in dynamic environments is needed. The abovementioned aim requires the establishment of a commonly accepted paradigm of the life-cycle (specifying, designing, developing and integrating) of the artefacts participating in this type of ecologies. In more detail, the

following steps are involved in the establishment of a well defined framework for the development of self-aware and self-reconfigurable symbiotic ecologies.

1. *Formulation and development of bio-inspired models and theories focusing on emergence, modelling cognitive and awareness processes, physical growth, and ontogenesis.*

These will help to design, construct and experiment with the interacting basic building blocks that will constitute the aimed ecologies. Clearly, a fostering research and synergistic work is needed between a broad range of scientific fields such as cognitive/experimental neuroscience, cognitive/developmental psychology, biological cybernetics, neuroinformatics and IT communities. A bio-inspired conception must be adopted based on the natural laws of evolution, survival and reproduction. Models relating energy, perception, computation, local interactions and interface plasticity must be studied in order to help in a counterbalance of low-energy consumption and sufficient capacity of extraction of meaning. This will pave the way for the generation of theory and technology of synthetic phenomenology (of the resulting ecology) meaning the understanding of the own self and its relation with the surrounding world. The accomplishment of these tasks involves i) the development or the refinement of dynamic bio-inspired specification models that describe how local behaviour becomes global and how to control it/reverse it or even how global strategies transform into local ones, ii) the specification of the morphology, "primal instincts" and limited capabilities of the artefacts inspired by the natural organisms that enable them to interact and cohabit with the others, iii) the specification of a minimal set of building blocks having certain physical properties and exhibiting certain behaviours, which will allow coherent society development and also the control of the communication between the participating individuals so that they can develop a desired behaviour and capabilities through their interaction with the environment.

2. *Development of a methodology on how to construct autonomous entities with "flexible structuring of the internal and external stimuli" and an integration framework that leads to the realization of self-reconfigurable ecologies.*

The techniques must be focused on intelligent periphery, morphology and possibly materials, inspired by the wide range of intelligent adaptations in non-human (neural) systems, gathering and exploiting knowledge about the world and the tasks, "environment models" used to codify world/task knowledge. By allying theoretical aspects such as the abovementioned to more practical bio-inspired technology driven aspects, the aim is to obtain a large understanding of social systems that learn from observing and from interacting with other more advanced systems. Key issues that have deep functional and economic significance for the design, construction, and maintenance of emergent ecologies include i) learning and evolution in an embodied artificial system; ii) autonomous self-construction and growth of artefacts ("epigenetic robotics"); iii) adaptation to the environment (possibly over several generations), and iv) robustness in performance. The tight multi-disciplinarity naturally addresses issues such as the practical limits of intelligent systems, the essential

properties of networks and sensors, the emergence of the complexity phenomena and how to control and profit from complex systems.

3. *Design and implementation of ambient knowledge/experience repositories available as substitutes of social memory.*

This could also contribute to the higher level cognitive processes including self-awareness, learning, and adaptation. As the symbiotic societies are dynamically reconfigured aiming at the accomplishment of new tasks (targeting to satisfy a higher goal), their formation heavily depends not only on space and time but also on the context of previous local interactions, previously configured teams, successfully achieved goals or failures. This means that in order to initially create, manage, communicate with, and reason about, such kinds of emergent ecologies, we need somehow to model and embed to these entities social memory, enhanced context memory, and shared experiences. These models should provide the appropriate abstractions and mechanisms for i) context reuse and reasoning, ii) social memory representation, and iii) knowledge and experiences inheritance. One step to this end is the design and implementation of evolving multi-dimensional ontologies that will include both non-functional descriptions, and rules and constraints of application, as well as aspects of dynamic behaviour and interactions. A core ontology could be open and universally available and accessible; however, during the ecology life-time the core ontology may be evolved into higher goal, application and context specific one [7]. Hence, ontologies describing specific application domains could be proprietary. Emerging behaviour, in this context, might be considered as a result of interactions among heterogeneous, seemingly incompatible or non pre-defined entities. Moreover, all higher-level constructions could be inherently able to use all the knowledge they will be able to access.

4. *Development of the necessary tools that constitute a development environment.*

New innovative concepts must be introduced and must be supported by specialized tools integrated in a development environment. The emergent ecologies must be based on the notion of the “autopoietic machine” built from basic building blocks (or cells in terms of genetics) [18]. Therefore, tools that provide representation schemes from physical world to digital space, learning and evolution mechanisms, communication protocols, and interaction patterns must be implemented and integrated in a development environment. The development environment will support the creation, management, communication with, and reasoning about, the emergent ecologies. Furthermore, it will apply novel methodologies for engineering the autonomous entities at different granularity levels. New programming paradigms are necessary (e.g., subject-oriented programming with subjects that are born, have a life cycle, can diminish, and have internal goals and intensions).

5 Engineering Emergent Phenomena

An especially complex task is to model and build autonomous interactive entities that could form extelligent ecologies exhibiting corporal literacy and leading to a synthetic phenomenology approach. The task is additionally complicated by considering that the resulting ecologies will operate into a ubiquitous environment and will be driven by autonomy, local perceptions and interactions, emergence, and distributed functioning. An important aspect on this focus is that although the entities will not have explicitly represented models of the world or of the others the emergent ecologies will unfold coherent collective behaviour based only on the entities' own agenda of actions and their intrinsic inclination to preserve their own goals.

Realizing the potential benefits of the UbiComp applications populated by autonomous simplistic entities will require improvements in currently available technology platforms and a translational research paradigm from basic-research findings. Hence the driving force behind the whole idea focuses on the adaptation of concepts from complex biological systems and novel fabrication technology platforms to build truly innovative swarm robotic systems for emerging real-life applications. Technological challenges posed by this approach and the investigative methodology to overcome them are described below.

5.1 Basic building blocks development

As described above, several levels of abstraction are possible for the formulation of the basic building block. Immediately, the engineer strives with the questions on i) which should be the basic building block, ii) what structural and functional properties it should encompass, iii) how it could interact with the others, and iv) how it could be realised. From a technology development point of view, an essential plan is needed which initially centres about the basic building block and considers as such every self-sustained digital (h/w or s/w) artefact with certain functionality that can operate without the contribution of others. That type of artefacts could be robots with pre-defined specialised capabilities, or could be sensors, motors, computational sources etc. In both cases emergent ecologies are possible to be formed exhibiting capabilities not found in the individuals.

Undoubtedly, this "macro" or high level perspective does not deal with issues as the structural parts, the realisation approach or the interaction patterns of the basic building blocks. Rather, such issues could be studied by disciplines such as artificial neural networks [2], evolutionary robotics [20] and machine learning [19] concentrated on building intelligent control systems. On the other hand, the problem-solving concepts from social systems in nature could be analysed and adapted for technical applications by using simulations. Simulation environments could give the possibility to study artificial ecologies of bio-inspired entities to close the capability gap between natural information processing systems and human-made ones. Additionally, they could help to reveal fundamental interrelations between

rules for entities of intelligent ecologies and the resulting global behaviour. Instead, as the focal point remains at the range of the behaviours that a basic building block should manifest, then the engineering methodology and the development starting point should be driven by the primitive behaviours approach followed in robotics applications. According to this approach, the overall behaviour of the system becomes the emergent effect of the interaction with the environment and the coordination of the primitive behaviours.

5.2 Engineering emergent behaviour

In dynamic environments, an individual must be reactive, that is, it must be responsive to events that occur in its environment, where these events affect either the individual's goals or the assumptions, which underpin the procedures that the individual is executing in order to achieve its goals. However, what turns out to be hard is building a system that achieves an effective balance between goal-directed and reactive behaviour. Furthermore, as the construction of the individuals must be based on the development of primitive behaviours, the issues of how to select potentially the correct behaviours in different circumstances and how to resolve conflicts between them are raised. The primitive behaviours approach considers that all the (individual) behaviours run in parallel and depending on the stimuli of the environment some of them manifest themselves by enabling a suppression mechanism and taking control of the actuators. However, this technique requires a pre-defined and exhaustively tested set of implicit rules (usually encoded into finite state automata) of firing priorities. Thus, this technique does not scale well even in moderate number of primitive behaviours and it lacks learning even in very often tasks.

In order to apply the well-established primitive behaviours approach in swarm societies that can learn and evolve component-oriented principles and practices could be employed. Synthetic behaviour control mechanisms could be developed based on bio-inspired approaches like spiking neural networks. These behaviour control mechanisms responsible for the arbitration and/or the composition of the primitive behaviours could also be subject of learning and evolution. The individuals may exhibit varying behaviour – capable of perceiving/exploring their environment, selectively focusing attention, initiating and completing several tasks. The learning and evolution could be studied and investigated at both the individual and social levels. In this case, the focal point must be the components of behaviour control mechanisms. The outcome could contribute to a novel dynamic and adaptive architecture of swarm systems that exploits the global effects through local rules/behaviour.

5.3 Engineering collective behaviour

Developing a robust swarm system, capable of exhibiting emergent intelligent collective behaviour is a non-trivial task. The nature of the social/collective behaviour sought and the environment that allows efficient development of ecologies requires research. In building a

swarm system communication plays a pivotal role and this explains the profuse number of publications in this area.

A flexible and light-weight approach is the indirect (stigmergic) communication. The essence of stigmergy is that the individual modifies a local property of the environment, which subject to environmental physics, should persist long enough to affect the individual's behaviour later in time. It is the temporal aspect of this phenomenon, which is crucial for emergent collective behaviour (collaborative exploration, building and maintenance of complex insect nest architectures etc) in societies of ants, agents and robotics. Thus, the individuals could be provided with the proper periphery (actuators/sensors) enabling them to emit/perceive electromagnetic signals emulating thus the biological "quorum sense" signals. Such a quorum sense communication may be based on an application-specific vocabulary that will be encoded in the signal. The specifics of the temporal modulation aspect of this "quorum sense signal" will come from theoretical biology and existing simulation studies. Additionally, the frequency of the signal will be determined after studying the combined influence of the physical medium properties, the range and interference constraints, power requirements and the size of available hardware components.

6 Summary

As everyday objects are being enhanced with sensing, processing and communication abilities, the near future of our everyday living/working is indicated by a high degree of complexity. The emergent complexity concerns the machine-machine and human-machine interactions as well as the provided services aimed at end users and at other machines. Into this rapidly changing AmI environments new requirements and research issues arise, and the need for a conceptual and analysis framework is apparent. This work attempts to introduce a bio-inspired word model that draws features from natural systems and applies them into symbiotic ecologies inhabited by both humans and artefacts. Furthermore, it introduces a high-level architecture of AmI spaces that encloses the fundamental elements of bio-inspired self-aware emergent symbiotic ecologies.

References

1. Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.
2. Arbib, M. A., *Handbook of Brain Theory and Neural Networks*, MIT Press, MA, 1995.
3. Bonabeau E., Dorigo M., and Theraulaz G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.
4. Brooks, R. A. Intelligence Without Reason. In *Proceedings of 12th Int. Joint Conf. on Artificial Intelligence*, Sydney, Australia, August 1991, pp. 569—595, (1991).
5. Brooks, R. A. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

6. Bullock, S., and Cliff, D., *Complexity and emergent behaviour in ICT systems*, HP Labs report HPL-2004-187, 2004.
7. Christopoulou E., Goumopoulos C., Zaharakis I., Kameas A., "An Ontology-based Conceptual Model for Composing Context-Aware Applications", *Workshop on Advanced Context Modelling, Reasoning and Management, Sixth International Conference on Ubiquitous Computing* (*UbiComp04*), Nottingham, England, 7-10 September 2004.
8. Digital Home Working Group, http://www.dhwg.org
9. Disappearing Computer initiative: http://www.disappearing-computer.net/
10. e-Gadgets project website: http://www.extrovert-gadgets.net
11. ISTAG website: http://www.cordis.lu/ist/istag.htm
12. Kameas A., Mavrommati I., Ringas D., Wason P., "eComp: an architecture that supports P2P networking among ubiquitous computing devices", in *Proceedings of the IEEE 2nd P2P Conference*, pp 57-64, Linkoping, Sweden, 5-7 September 2002
13. Kameas A., Bellis S., Mavrommati I., Delanay K., Colley M., Pounds-Cornish A., "An architecture that treats everyday objects as communicating tangible components", in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, (*PERCOM2003*), Texas-Fort Worth, 2003.
14. Kameas A., Mavrommati I. and Markopoulos P., "Computing in tangible: using artifacts as components of Ambient Intelligent Environments". In *"Ambient Intelligence: The evolution of Technology, Communication and Cognition"*, (G. Riva, F. Vatalaro, F. Davide and M. Alcaniz, eds), IOS press, pp 121-142, 2004.
15. Kennedy, J. Eberhart, R. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.
16. Lund, H-H., Marti, P. Physical and Conceptual Constructions in Advanced Learning Environments. *Interaction Studies Journal*, 5(2): 271-301, 2004.
17. Lund, H. H. Neural Building Blocks. In *1st International IEEE EMB Conference on Neural Engineering*, Capri, Italy, March 20-22 2003. IEEE Press.
18. Maturana, H. and Varela, F., *Autopoiesis and Cognition: The realization of the living*, D.Reidel, Boston, 1980.
19. Mitchell, T. M., *Machine Learning*, McGraw-Hill, 1997.
20. Nolfi, S. and Floreano D., *Evolutionary Robotics*, MIT Press, MA, 2000.
21. Shoham, Y. Agent-oriented Programming, *Artificial Intelligence*, 60(1):51-92, 1993.
22. Stewart, I., Cohen, J. *Figments of Reality: the origins of the curious mind.* Cambridge University Press, 1997.
23. Wooldridge, M., Intelligent Agents, in G. Weiss (ed.) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, pp. 27-77, 1999.

A Semantic Choreography-driven Frequent Flyer Program

José-Manuel López-Cobo* and Alejandro López-Pérez* and James Scicluna**
*Atos Origin SAE, Albarracin 25, Madrid, Spain
**University of Innsbruck, Digital Enterprise Research Institute, Technikerstrasse 21a, 6020 Innsbruck, Austria

Abstract. *Frequent Flyer Programs reward their members with Travel Services from affiliated Service Providers. In this paper, we present a semantic application performing a Frequent Flyer Program in which the customers can create and reuse travel packages. The application is built upon a Service Oriented Architecture, accessing, discovering, composing and invoking Semantic Web Services for the management of the Travel Packages. The composition of semantic services is driven by Choreography, using the Web Service Modelling Ontology (WSMO) as a framework to describe both the service capability and the Service behavior.*

Keywords: Semantic Web Services, Case Studies, Choreography, Frequent Flyer Programmes

1 Introduction

Loyalty marketing has become a key element in many marketing companies, because it has been proven to be measurable, and considerably effective. In fact, many companies have changed their marketing strategy from mass-market advertising to different direct-marketing strategies, like loyalty-marketing. Frequent flyer programs (FFP) are the most visible and the most highly developed of the award programs. They are in fact a subset of a larger class of related marketing approaches known as frequency marketing, relationship marketing or loyalty marketing. Frequent flyer programs are loyalty-marketing programs [ffp]. STREAM Airlines is a fictitious airline company of the fictitious STREAM Group. The STREAM Group has a FFP called STREAM Flows! System (SFS in short). The customers of the SFS (by means of a membership) can obtain travel points from purchasing services of the STREAM Group. These services might be an airline ticket, a hotel booking, a car rental or a shuttle booking (amongst others). There are several companies affiliated to SFS. Each service of the program has its counterpart in points and these points can be consolidated and exchanged with one or more services. With respect to the cost of the FFP, a division can be made from the IT perspective and the rest. In particular, for a consulted FFP of a medium-size European airline company the possible costs might include:

- IT costs such as software development, human resources, hardware and facilities: €3.5 Million per year

- Communication such as to create and send material to the customers and retain their loyalty: €1 Million per year
- Organization and Marketing such as the cost for maintaining the business and all the relationships with the partners of the program: €1.5 Million per year
- Point cost such as those stored in the FFP. In that sense, the FFP acts as a bank, a voucher for the points from the customers: €36 Million per year.

For us it's quite clear that solutions like SFS can allow FFPs to decrease their IT cost, making them more profitable due to the increasing incomes from advertisement and marketing.

SFS allows customer and Service Providers to collaborate with each other to reduce the cost of the system and share information. Using this paradigm, the share of knowledge and the understanding from the machine point of view arises as a necessity. The Semantic Web comes at this point as one possibility to reduce the gaps between the ability of the customer and the operability of Service Providers. SFS provides a catalogue to their members where they can search and browse travel services. It also provides mechanisms to create packages of travel services (packages which can be stored and reused). Each travel service contracted is paid using the points gathered in previous transactions with the SFS.

Within the EU funded INFRAWEBS project, we have faced the challenge to build a Semantic application performing a Frequent Flyer Program for a fictitious airline company. The architecture of the application follows the guidelines of the project and it is based on a Service Oriented Architecture over a P2P paradigm as shown in Figure 1. In this architecture, each peer has the same infrastructure and implements the same set of components (SWS Designer, SWS Composer, SWS Executor, Organisational Memory (OM), Semantic Information Router (SIR), and Distributed Repository among others).

Semantic interoperability has been sought in this application. For that reason we have used the framework proposed by WSMO [RLK05], paying special attention to the WSMO choreography and orchestration [SPR05] to show the interaction between the customer and the Service Provider and within a Service Provider. The INFRAWEBS architecture is based on two different stages: Design and Runtime. Figure 1 shows the INFRAWEBS overall architecture organized by stages. Some of the components belong to more than one stage, because they offer features that are used in more than one stage. Consequently, we have put them at the boundary of both stages, thus making them shared.

At Design/Development stage all the activities offer components for annotating, designing and composing Semantic Web Services. At the Runtime stage there are the execution, monitoring and runtime-discovery components to represent the core group. All of these components are briefly described below.

The Functional Architecture of the Designer is a set of functions organized semantically as modules, which are needed for designing a semantic web service using a selected framework for service description. According to the above mentioned basic design principles, a semantic service created by INFRAWEBS SWS Designer will be described in WSMO.

SWS-Executor is responsible for executing Semantic Web service descriptions based on WSMO. At the heart of this module are the choreography and orchestration engines which execute the rules defined in the interface descriptions. On top of the executor is also a Quality of Service Broker (QoS Broker) responsible for monitoring and calculating metric data of the Web services that are being executed.

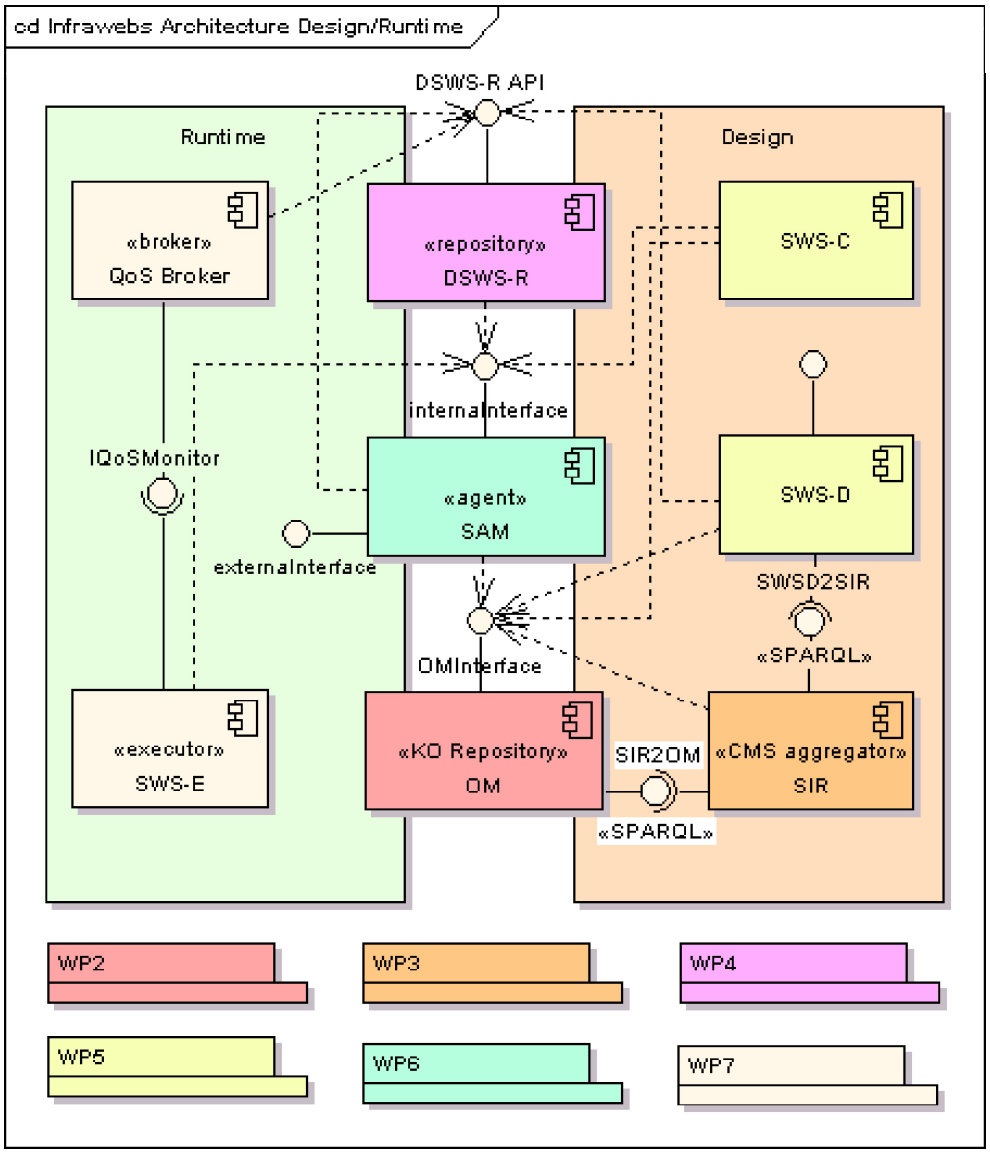


Figure 1: Infrawebs Architecture

OM (Organizational Memory) contains non-logical representation of the WSMO objects stored in DSWS-R (so called ”knowledge objects”) as well as some additional non-semantic data e.g. graphical models of SWS and ”natural language” templates for WSMO Goals needed for modifying or using these objects. OM is a Web service implementation of a case-based memory.

SIR is a meta-data based content management and aggregation platform, that will have a SPARQL [PS06] query interface, which will be used by the SWS-D to query for service description meta-data, and the UDDI interface integration with UDDI based service repository.

DSWS-R Repository is designed to enable efficient storage and retrieval of WSMO SWS descriptions. It is realized as an extension of the wsmo4j and ORDI components, which already provide basic WSMO software infrastructure. The DSWS-R API defines the interface methods that are available to the other INFRAWEBS components.

SAM (Service Access Middleware) provides a retrieval and execution interface for advertised SW services. The user/customer mandates a user interface agent for fulfilling the service demand. The user/interface agent gives recommendations based on the user's query.

The rest of the paper is structured as follows: in Section 2 a scenario will be described for the creation of a package using the application, in Section 3 we will describe the WSMO elements we have created for this application, emphasising on the Ontologies needed for describing the domain and the application. In Sections 4 and 5 we will show how this application faces the discovery and composition of travel services, highlighting the orchestration and choreography needed both from the requester and provider point of view. We will finally wrap up with some related work and conclusive remarks in Sections 6 and 7 respectively.

2 Scenario Description

In this section we will describe a typical scenario in which a customer uses the SFS portal for the creation of a package of Travel Services, find suitable services for each slot of the package and, finally, contract the proposed services for booking a flight, a hotel room and the renting of a car.

1. The customer logs in the application. We will define a scenario in which *a customer ”Jane Doe”, having 2500 points in her account wants to go to Madrid from Brussels and rent a car in Madrid Airport (Barajas) to go to Toledo and book a room in a 4 stars hotel in Toledo”*.

2. The customer defines her desires on the ”Voyage Planner”. Within the planner, Jane can choose a stored package (flight + hotel + rent-a-car with standard restrictions) or create a new one. Jane will create her own package. A box will appear and she will drag and drop icons from the left (one for the flight, one for the hotel and one for the car). Each slot in the box represents an abstract service (abstract in terms of desired behaviour but not fully specified [Pri04]). The definition of the constraints may be based on time, distance, accumulated amount of points and whatever other property with an ordinal and quantitative metric. Any of these properties can be graphically defined as connections between the slots. For her own package she defines the *following set of constraints*:

   - The location parking of the rent-a-car service has to be within 3 km of the destination airport.

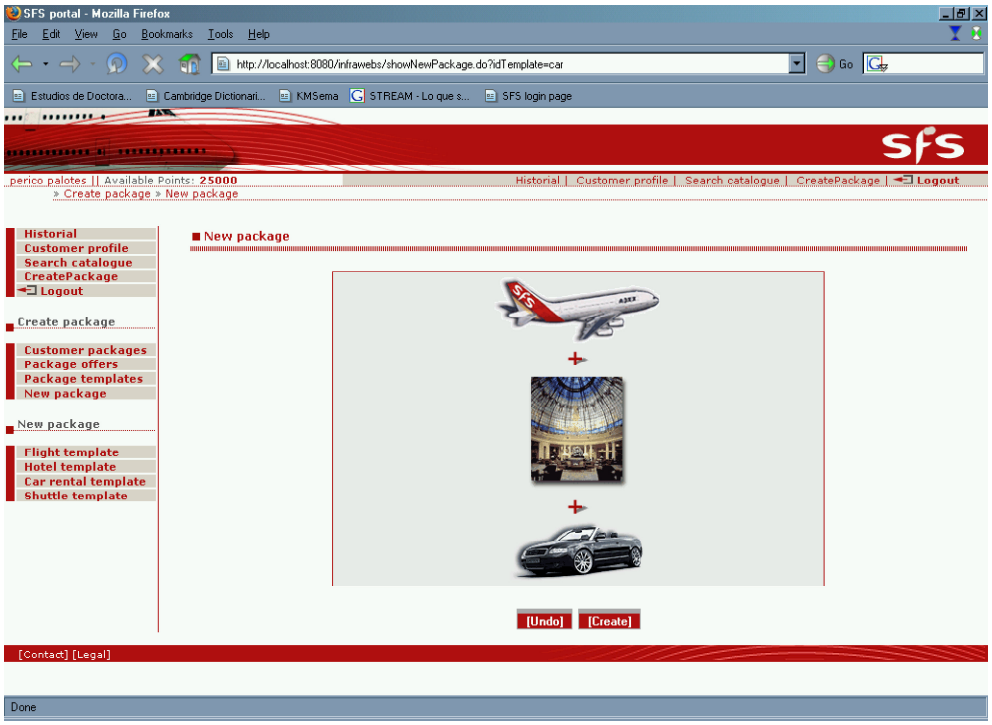


Figure 2: Screenshot of the creation of a package by means of SFS

- The rent-a-car service must be open at the scheduled landing time.
- The hotel check-in date has to be later than the scheduled time for landing.
- The accumulated amount of points for the three services can not be greater than 2200 points.

3. Once she has the package with the three slots filled, a WSMO goal is created using the templates and the values obtained from the customer interaction with the interface. These goals are sent to the Discovery Component and the Discovery searches into the SFS Catalogue for services that are available for each slot.

4. The selection phase not only performs selection task but also checks the coherence between the services chosen (the set of services chosen, as a whole, have to fulfil the constraints for the package imposed by the customer).

5. The customer invokes the services which fulfil her package and each Service Provider will contact SFS for the payment of the service (the payment, from the customer perspective is made with points, but from the Service Provider perspective is made by the SFS). This arrangement between SFS and Service Providers is out of the scope of this paper.

3 Ontological Structure of the SFS

In this section we describe briefly the SFS ontologies that model the domain presented in our scenario, and which will be used by the Semantic Web services developed by the Service Providers. These ontologies have been written in the Web Service Modelling Language (WSML) [dBLK+05] which is the base language of the WSMO Conceptual Model.

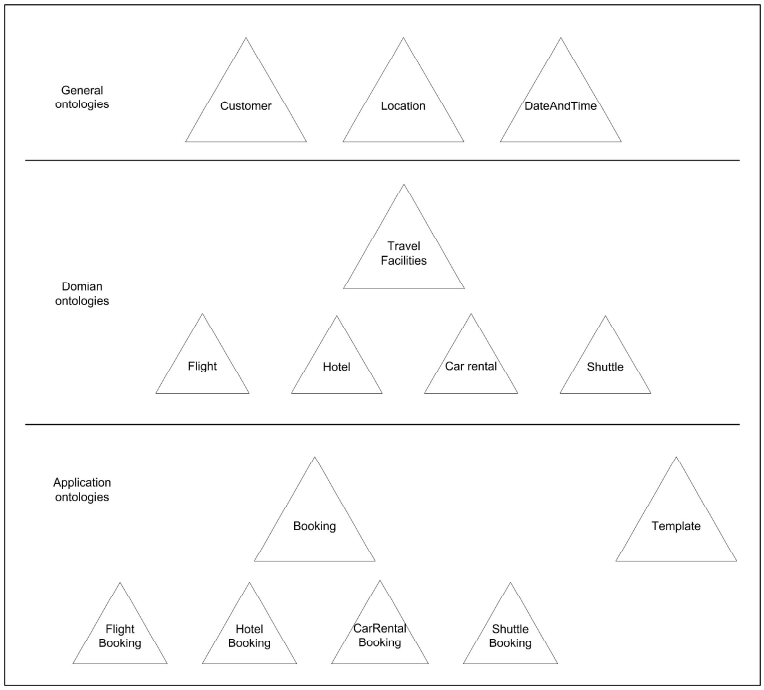


Figure 3: SFS Ontologies

According to the classifications of Van Heijst et al. in [vHSW97] and Mizoguchi et al. in [MvWI95], we can distinguish between the following types of ontologies:

- General ontologies, which represent common sense knowledge reusable across domains.
- Domain ontologies, which represent reusable knowledge in a specific domain.
- Application ontologies, which represent the application-dependent knowledge needed.

We will now describe in more detail some of these ontologies, paying more attention to those that are related to the templates and packages. In the following sections we will use them to explain how SFS makes use of the template and the packages for discovering, selecting, composing and invoking Semantic Web Services. Some of the ontologies used in this paper will be only named. For a complete overview of the SFS Ontologies, we refer the reader to [LCLPP05].

3.1 Domain Ontologies

These ontologies describe domain concepts used in the SFS application. Amongst these ontologies are Tourism Service Providers and the Services's Ontologies (flight, hotel, rent-a-car, shuttle). The service providers will offer different flights, hotel rooms, cars, shuttles and they will have to store the reservations already made in order to be able to make future bookings. The ontologies also include taxonomies such as the rooms of a hotel.

3.2 Template and Package Ontologies

This ontology describes the template and package concepts. It also describes the package constraints. The ontology will be used to find the necessary goal templates for building the package. Logically, the template and the package template are stored in the SFS as instances of these concepts. Each instance will have a file (for a template) or many (for a package) which represent the abstract capability that the requester wants to achieve. The system will build the goal (for a service or for "many" services) using these templates and the inputs from the customer (using a rich graphical interface or even natural language in future versions). The example below describes a goal template for booking a hotel room (Listing 1).

```
namespace {_"http://www.wsmo.org/goals/sfs/hotelRoomBookingGoalTemplate#",
    bo _"http://example.org/sfs/bookingOntology#",
    hb _"http://example.org/sfs/hotelBooking#",
    cu _"http://example.org/customer#",
    dc _"http://purl.org/dc/elements/1.1#",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#"}

goal _"http://www.wsmo.org/goals/sfs/hotelRoomBookingGoalTemplate"

capability hotelRoomBookingGoalTemplateCapability

  postcondition
    definedBy
      ?HotelBooking[
        hotelStay hasValue ?BookingRequest,
        buyer hasValue ?Buyer
      ] memberOf hb#hotelRoomBooking and
      ?BookingRequest[
        checkIn hasValue ?CheckIn,
        checkOut hasValue ?CheckOut,
        numberOfPersons hasValue ?Amount,
        hotelStars hasValue ?Stars,
        numberOfBeds hasValue ?NumberOfBeds,
        smoking hasValue ?Smoking
      ] memberOf hb#hotelStay and
      ?Buyer[
        email hasValue ?Email,
        name hasValue ?Name,
        surname hasValue ?Surname
      ] memberOf cu#customer.
```

Listing 1: Goal Template for booking a Hotel Room

3.3 Booking Ontologies

The booking ontology describes the domain of booking services (which is a specialization of the purchase of an item) within a B2C scenario. The main constructs of the booking ontology are: **Booking**: the overall construct that holds all aspects for a booking where a buyer reserves a service from a service provider, **Booking Partners**: the parties involved in a booking, i.e. buyer and seller, **Payment**: specifies notions of payment, **Delivery**: specifies delivery methods for delivering a booking ticket from the seller to the buyer

4 Discovery Related Aspects of Templates and Packages

The ability to build packages of Travel Services is one of the main features of the SFS Application. The whole idea of the system is to provide the customers the ability of build packages to pack services into a whole service for reusing and pricing purposes, reducing the interaction with humans in the Provider side of the program, letting the customer the responsibility of the decision.

4.1 Abstract Services vs. Concrete Services

The SFS administrator will publish templates in the Catalogue. These templates are in the form of an Abstract Service Description (ASD) [Pri04]. An ASD is the scaffolding used by the requester to build the goal which represents the desire of the customer when she wants to obtain some Service (or some of them) to perform her need. When the customer chooses a template the ASD linked to the template is incomplete. The costumer will fill the ASD with constraints. These constraints will form part of the axioms and functions of the requested capability. Once she has the package with the slots filled, a WSMO Goal is created using the templates and the values obtained from the customer interaction with the interface. These goals are sent to the Discovery Component which searches into the SFS Catalogue for available services that can fulfil each slot. The following goal Template is selected when the customer wants to book a room in a hotel:

Desire : The booking of a room in a hotel and the delivery of the ticket to the buyer via email.

Postcondition : a hotel room booking for a buyer (the SFS customer), the payment method chosen will be by points.

Effect : delivery of the hotel room booking ticket via email.

4.2 Discovery and Selection

The customer will send a query to the Discovery module to obtain the set of Services which better suits the requirements of the Customer. For each slot belonging to the package, the Discovery module will be asked to find a set of Services that performs the desired behaviour, maintaining the integrity with respect to the constraints of the slot and checking that the package constraints are satisfied. The process for the selection is as follows (the interaction of the customer with the Discovery has been depicted as a choreography interface in Figure 4):

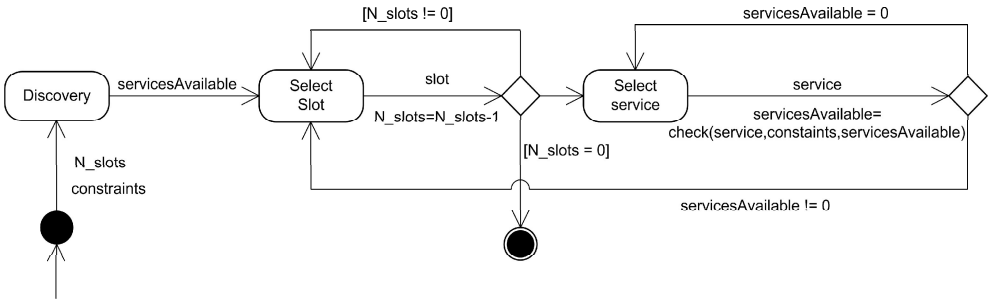


Figure 4: Process of Backtracking and Selection phase

- The user receives a list of possible results for each slot She can manually select one service or she can be advised by the system (using QoS metrics or other customer preferences defined previously).
- The selected service is asked for final offers which can fulfil the customer preferences and constraints (the service may have more than one possible realisation of the service for a concrete goal). These offers are specific instantiations of the service. If an offer is finally accepted by the customer, the concrete invocation endpoint of the service is stored until all the services are chosen.
- When all the slots are concretized with final offers, then the package is ready to be enacted. Since all the services have agreed with the overall constraints, the order for the enactment is not an issue and they can be contracted in whatsoever order and time (even in parallel).

5 Composition and Enactment of Services

We have defined Web Services for booking a flight, a hotel room or renting a car. These Web Services can be composed of other Web Services. This setting allows modelling all notions of a WSMO Web Service description: a Capability of the end-user service and its Choreography for user-service interaction, as well as the orchestration which incorporates the aggregated Web Services. WSMO Choreography deals with interactions of the Web service from the client's perspective. We base the description of the behaviour of a single service exposed to its client on the basic ASM models: (1) they are **state-based**, (2) they represent a state by a **signature**, and (3) it models state changes by **transition rules** that change the values of functions and relations defined by the signature of the algebra. Taking the ASM methodology as a starting point, a WSMO choreography consists of the following elements:

- State: A state is described in terms of non-empty set of ontologies.
- Guarded transitions: Transition rules that express changes of states by changing the set of instances. These rules are expressions that take the form of: **if** *Cond* **then** *Updates*, **forall** *Variables*

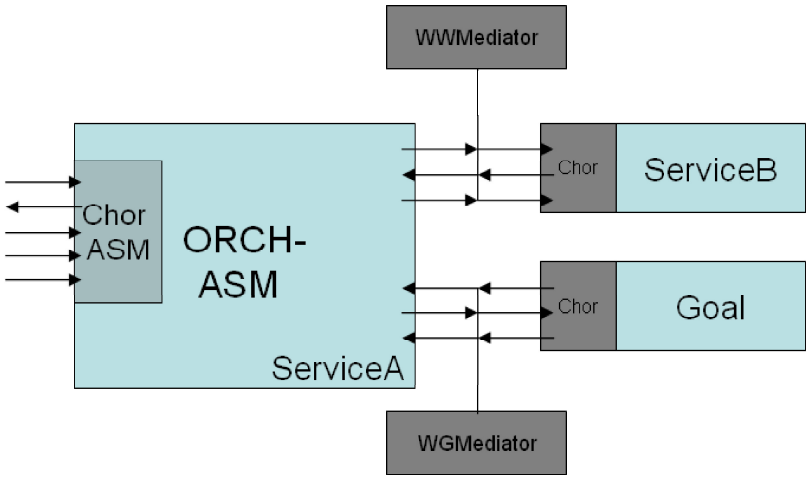


Figure 5: WSMO Choreography and Orchestration Interfaces

with *Cond* **do** *Updates* (for universal quantification) and **choose** *Variables* **with** *Cond* **do** *Updates* (for non-deterministic choices)

- The Orchestration part of a service interface defines how the overall functionality of the service is achieved in terms of the cooperation with other service providers. It describes how the service works from the provider's perspective (i.e. how a service makes use of other WSMO services or goals in order to achieve its capability, see Figure 5)

We will detail the elements that define a Web Service, highlighting the interface of the Web Service and, especially the Ontologies used for describing the choreography and the orchestration. We use the example of the Hotel Room Booking Web Service . This web service is provided by MH hotel service provider. It offers 4 stars hotels in Spain.

Functionality : A booking for a hotel room will be created and the reservation ticket will be sent to the buyer's email.

Capability :

- *Assumption*: The required payment method is defined. In this case it will be by the subtraction of points from the FFP membership.
- *Precondition*: The required inputs for this service are
  1. The buyer contact information, including the e-mail address
  2. the buyer hotel room booking preferences

- *Postcondition*: A hotel room booking for a buyer is performed and the payment for this room will be made with points. The hotel provider is identified.
- *Effect*: The delivery of the hotel room reservation is done by email to the buyer.

For the sake of clarity, the semantic description is split in two separate listings: one for the capability (Listing 2 and the other for the interface description (Listing 3). In both cases the same namespace is assumed to be used. The assumption in Listing 2 is omitted since the reduction of the points from the system is out of the scope of the SFS scenario.

```
namespace {_"http://example.org/sfs/services/MHHotelRoomBookingWS#",
    bo _"http://example.org/sfs/ontologies/bookingOntology#",
    hb _"http://example.org/sfs/ontologies/hotelBooking#",
    cu _"http://example.org/customer#",
    dc _"http://purl.org/dc/elements/1.1#",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#"}

webService _"http://example.org/sfs/services/MHHotelRoomBookingWS"

capability MHotelWSCapability

    sharedVariables{?BookingRequest, ?Buyer}

    precondition
      definedBy
        ?BookingRequest memberOf hb#hotelStay and
        ?Buyer memberOf cu#customer.

    postcondition
      definedBy
        ?BookingRequest[
          checkIn hasValue ?CheckIn,
          checkOut hasValue ?CheckOut,
          numberOfPersons hasValue ?Amount,
          hotelStars hasValue ?Stars,
          numberOfBeds hasValue ?NumberOfBeds,
          smoking hasValue ?Smoking
        ] memberOf hb#hotelStay and
        ?Buyer[
          email hasValue ?Email
        ] memberOf cu#customer implies
          exists {?HotelBooking} (
            ?HotelBooking[
              hotelStay hasValue ?BookingRequest,
              buyer hasValue ?Buyer
            ] memberOf hb#hotelRoomBooking and
            ?BookingRequest[
              checkIn hasValue ?CheckIn,
              checkOut hasValue ?CheckOut,
              numberOfPersons hasValue ?Amount,
              hotelStars hasValue ?Stars,
              numberOfBeds hasValue ?NumberOfBeds,
              smoking hasValue ?Smoking
            ] memberOf hb#hotelRoomBooking and
            ?Buyer[
              email hasValue ?Email
            ] memberOf cu#customer).
```

```
effect
  definedBy
    ?HotelRoomBooking [
      bookingIdentifier hasValue ?BookingIdentifier,
      bookingTicket hasValue ?HotelRoomBookingTicket,
      buyer hasValue ?Buyer,
      seller hasValue ?MHotelServiceProvider
    ] memberOf hb#hotelRoomBooking implies
      exists {?OnlineDelivery}
      (?OnlineDelivery [
        deliveryItem hasValue {?HotelRoomBookingTicket},
        sender hasValue ?MHotelServiceProvider,
        receiver hasValue ?Buyer,
        onlineDeliveryMethod hasValue "email"
      ]memberOf bo#onlineDelivery).
```

Listing 2: Capability Description for the Hotel Booking Web Service

Interface :

- *Choreography*:
  - When a hotelBookingRequest is received, if it fulfils the service provider constraints, all the hotel-room combinations will be returned to the customer (using the findHotelStay-Mediator).
  - Once the customer selects the best combination, and the hotelStayInfo is checked, a hotelBookingInstance is completed with all the attributes filled in.
- *Orchestration*: When a hotelBookingRequest is received, the service must make use of another service which provides the HotelStay (this internal Web Service looks into the information space of the provider and returns instances of the concept HotelStay)

The behaviour of the Interface of the Web Service is depicted in 6 below.

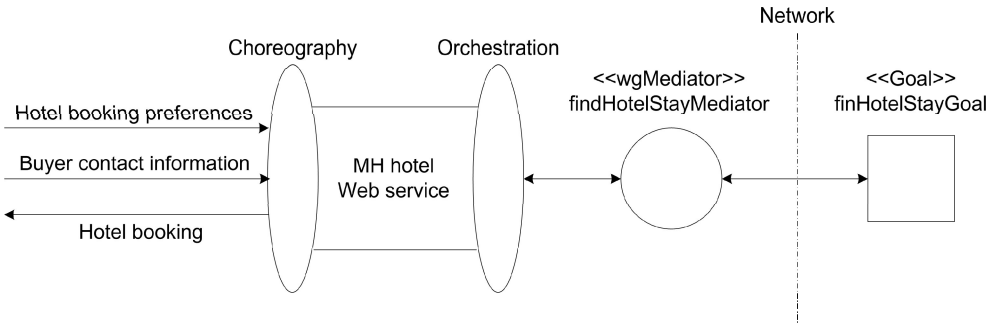


Figure 6: MH Hotel Web Service accepts the Hotel Booking preferences and the buyer contact information and returns the booking to the customer

The Interface of the WSMO Web Service is shown in Listing 3. Note that we omit descriptions such as non-functional properties to keep the document concise. Furthermore the orchestration description is not shown since this is still ongoing work within the WSMO community.

```
interface MHotelWSInterface
  choreography
    stateSignature

      importsOntology{
        _"http://www.wsmo.org/ontologies/sfs/hotelBooking",
        _"http://example.org/customer"
      }

      in
        hb#hotelStay withGrounding _"http://example.org/MHotelWS#wsdl.interfaceMessageReference(
              MHotelServicePortType/HotelRoomBooking/In)",
        cu#customer withGrounding _"http://example.org/MHotelWS#wsdl.interfaceMessageReference(
              MHotelServicePortType/HotelRoomBooking/In)"

      out
        hb#hotelRoomBooking withGrounding _"http://example.org/MHotelWS#wsdl.interfaceMessageReference(
              MHotelServicePortType/HotelRoomBooking/Out)"

    transitionRules

      forall {?HotelRoomBookingRequest} with
        ((?HotelRoombookingRequest[
            hotelCity hasValue ?city,
            checkIn hasValue ?checkIn,
            checkOut hasValue ?checkOut,
            hotelStars hasValue ?stars,
            allowPets hasValue ?allowPets,
            allowGroupsReservation hasValue ?allowGroupsReservation
        ] memberOf hb#hotelRoombookingRequest)
        and (
          exists ?MHotelServiceProvider
            (?MHotelServiceProvider[
                hasCompanyID hasValue "MH Hotels",
                availableCities hasValue ?city,
                allowPets hasValue _boolean("false"),
                allowGroupsReservation hasValue _boolean("true"),
                minStars hasValue 4,
                hotels hasValue {?Hotel}
            ]memberOf hb#HotelServiceProvider)
          )
        ) do
        add(_# memberOf hb#hotelBooking)
      endForall

      forall {?HotelStayRequest} with
          (?BookingRequest[
              checkIn hasValue ?CheckIn,
              checkOut hasValue ?CheckOut,
              numberOfPersons hasValue ?Amount,
              hotelStars hasValue ?Stars,
              numberOfBeds hasValue ?NumberOfBeds,
              smoking hasValue ?Smoking
            ] memberOf hb#hotelStay and
            ?Buyer[
```

```
            email hasValue ?Email
          ] memberOf cu#customer) do
        add(_#[
            hotelStay hasValue ?BookingRequest,
            buyer hasValue ?Buyer
          ] memberOf hb#hotelBooking)
    endForall
```

Listing 3: Interface Description of the Hotel Booking Web Service

6 Related Work

Most of the related work in this area has been produced by the WSMO Community in terms of Use Cases. One of these Use Cases relates to Amazon [KRS05] which follows a bottom-up approach for providing semantics to the Amazon Web Service. First, an ontology which describes the data types used by the Amazon WSDL file is defined. The capability and interface descriptions use the concepts in this ontology to describe what the service provides and how the client can interact with the service. The Virtual Travel Agency (VTA) Scenario [SLL+04] is a similar approach to the Stream Flows System depicted in this paper. However, the VTA scenario does not describe the interaction of the client with the service by means of a choreography description. Within the DIP integrated project [dip] various case studies have been carried out. These include: a Business to Business Telecommunications scenario acitedip:d83, e-Government [DD05] and also an eBanking study [LCC+05].

7 Conclusions and Further Steps

In this paper we have presented an application for a Frequent Flyer Program. It takes advantages of the Semantic Web, particularly of Semantic Web Services. Goal templates ease the usage of the system for the customer since they hide the complexity of the logical descriptions of WSMO. A choreography-driven selection phase has been proposed for this kind of composite services (packages). The composition and invocation of composite services has been also detailed, making use of the orchestration and choreography proposed in WSMO. Our next steps for the application is the improvement of the description by means of mediators, taking advantage of the different kinds of mediators which are supported by WSMO (mediators for not only data mismatching in terms of ontologies but also for the interconnection of goals and web services and among them). Finally, we have created the INFRAWEBS process, a software development driven methodology that will guide all the partners to integrate the INFRAWEBS components in the framework. The final implementation of the SFS system, scheduled for this year, will instantiate this INFRAWEBS Integrated Framework. Furthermore, we plan to improve its features including security aspects and the inclusion of tools for the graphical creation and management of Goals templates based on the SWS Designer of the INFRAWEBS Project.

Acknowledgments We would like to thank the aid of the INFRAWEBS team for their fruitful discussions concerning this use case. This work is supported by the EC funded IST project INFRAWEBS (FP6-511723).

References

[dBLK+05] Jos de Bruijn, Holger Lausen, Reto Krummenacher, Axel Polleres, Livia Predoiu, Michael Kifer, and Dieter Fensel. The Web Service Modeling Language WSML. Technical Report D16.1, DERI Innsbruck, 2005.

[DD05] Rob Davies and Christian Drumm. Case Study eGovernment: Prototype Requirements Specification. Technical Report D9.2, DIP Integrated Project, 2005.

[dip] Data, information and process integration with semantic web services.

[ffp] Frequent flyer.com and frequent flyer marketing.

[KRS05] Jacek Kopecky, Dumitru Roman, and James Scicluna. Wsmo Use Case: Amazon E-Commerce Service. Technical Report D3.4, DERI Innsbruck, 2005.

[LCC+05] Silvestre Losada, Oscar Corcho, Jesús Contreras, Mónica Martínez Montes, José Luis Bas, Sergio Bellido, Richard Benjamins, and Jordi Ribas. Case Study eBanking: WSMO Descriptions of Application. Technical Report D10.4, DIP Integrated Project, 2005.

[LCLPP05] José-Manuel López-Cobo, Alejandro López-Pérez, and Clara Pezuela. Requirements Profile and Knowledge Objects. Technical report, Atos Origin SAE, 2005.

[MvWI95] Riichiro Mizoguchi, Johan van Welkenhuysen, and Mitsuru Ikeda. Task Ontology for Reuse of Problem Solving Knowledge. In *Second International Converence on Building and Sharing of Very Large-Scale Knowledge Bases (KB&KS 1995)*, Twente, The Netherlands, 1995.

[Pri04] Chris Priest. A Conceptual Architecture for Semantic Web Services. In *Third International Semantic Web Conference (ISWC 2004)*, November 2004.

[PS06] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Technical report, World Wide Web Consortium, 2006.

[RLK05] Dumitru Roman, Holger Lausen, and Uwe Keller. Web Service Modeling Ontology (WSMO). Technical Report D2, DERI Innsbruck, 2005.

[SLL+04] Michael Stollberg, Holger Lausen, Ruben Lara, Uwe Keller, Michal Zaremba, Armin Haller, Dieter Fensel, and Michael Kifer. WSMO Use Case: Virtual Travel Agency. Technical Report D3.3, DERI Innsbruck, 2004.

[SPR05] James Scicluna, Axel Polleres, and Dumitru Roman. Ontology-based Choreography and Orchestration of WSMO services. Technical Report D14, DERI Innsbruck, 2005.

[vHSW97] G. van Heijst, A. Th. Schreiber, and B.J. Wielinga. Using Explicit Ontologies in KBS Development. In *International Journal of Human-Computer Studies*, volume 46, pages 183–292, 1997.

Characterization of Semantic Grid Engineering

Joachim Bayer*, Fabio Bella*, Alexis Ocampo*
* Fraunhofer Institute for Experimental Software Engineering (IESE)

Abstract. *Currently, there is confusion about processes to apply for engineering solutions based on semantic-rich descriptions of services. We present the results of a survey that aimed at identifying suitable engineering processes in this emerging domain.*

Keywords: Software Engineering Process, Software Engineering Lifecycle

1 Introduction

Although the basic concepts of service-oriented architectures and web services, on the one hand, and grid technologies on the other hand, have become very popular in the last few years and are recognized as enabler architectural styles for modern enter-prise and scientific applications, much confusion remains concerning strategies and processes suitable for engineering grid-enabled web services. Real projects indicate that many difficulties arise related to a concerted application of the practices and that suitable means for such coordination are urgent needed.

Existing modeling approaches such as object-oriented analysis and design, enter-prise architecture frameworks, and business process modeling represent widely-accepted, high-quality practices for identifying and defining appropriate abstractions within service-oriented solutions [11].

Concerning the adoption of grid technologies, many organizations do fully under-stand the value of virtualizing organizational resources and making them available in the form of services, yet it is still unclear where to start and which strategy to apply in order to minimize risks and provide real benefits. Things become even more difficult when grid services are coupled with the emerging semantic web services paradigm, which promises automated discovery, creation, composition, and enactment of such services. The idea behind handling services in a semantic-oriented way is based on formal semantic-rich descriptions of services and machines capable of processing the information contained in these descriptions. Engineering a solution in this area mainly means the following: a) the semantic annotation

and publication of services, b) the engineering of machines (e.g., agents) capable of autonomously coming up with a solution to a given problem. The latter implies several steps including, for example, a machine to break down the problem to be solved into smaller problems that can be solved through atomic services, discovering the needed services, orchestrating them for achieving one composed service suitable for solving the main problem, invoking the services discovered according to the sequence orchestrated, and eventually translating the results from some atomic services according to the semantic expected by other services for their input (e.g., translating an output in inches into an input in centimeters).
In this paper, we present a systematic investigation of the field of semantic grid together with its related fields and the results of a literature search aimed at determining currently available engineering processes. The paper is focused on the fields of web services, grid services, and semantic web services.
The remainder of this paper is structured as follows: Section 2 provides an over-view of the main fields related to semantic grid. Section 3 presents a practical example that illustrates how the same scenario can be implemented in different ways by applying web services, grid services, or semantic web services. Section 4 presents existing engineering approaches in the considered fields. Section 5 summarizes the paper.

2 Semantic Grid Domain

The semantic grid domain is an emerging domain and a stable reference terminology does not exist yet. In this section, we provide an overview of our understanding of the domain. The characterization does not aim at being a reference characterization but rather at improving the readability of the reminder of the article.
The field of semantic grid strongly relates to several other emerging areas of interest, such as grid services and semantic web services. Due to the novelty of many of the related fields, the relationships between them are not always clear and will be highlighted in the following. Fig. 1 gives an overview of the relevant fields. The arrows in the figure mean that a field is based on concepts and solutions provided by other underlying fields. The fields of grid computing, web services, and semantic web depicted at the bottom of the figure do not present mutual relationships, since they originate from different research areas and address different problems. They provide the concepts needed for the definition of the fields grid services and semantic web services, which are both characterized by the use of technologies related to web services. The field of semantic grid, which results from the idea of describing grid-enabled web services in a semantic-rich way, is shown at the top of the figure.
In the following, the fields are further discussed in terms of the main problems addressed, the organizations involved in their solution, and the approaches currently proposed.
Grid Computing - The idea behind the field of grid computing is to make use of unused resources such as CPU cycles, network bandwidth, or data storage for solving massive computational problems. To achieve the goal, heterogeneous sets of computers such as

desktop computers and supercomputers are treated as a virtual cluster embedded in a distributed environment. Furthermore, grid computing solutions support virtual organizations in sharing resources and data beyond the boundaries of administrative domains.
Research in this field began in the early days of computer networks and several different approaches have been proposed. [22] provides a brief overview of eminent projects such as Condor, CODINE, Legion, Nimrod, and UNICORE.

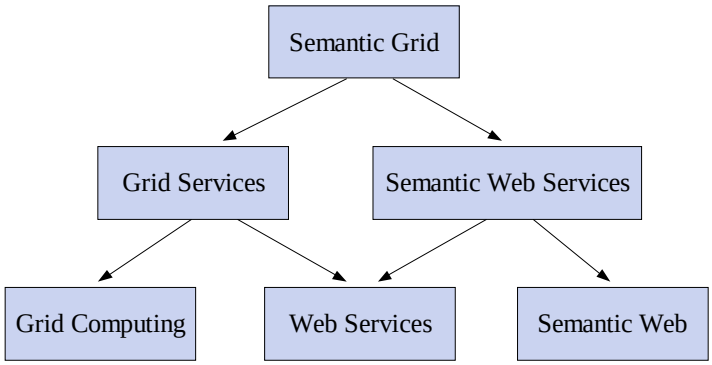


Fig. 1. Semantic grid and related fields

Web Services - Web service technology is used for exchanging data between applications. Its main goal is to enable the exchange of data over computer networks (like, for example, the internet) between heterogeneous software applications (i.e., applications written in various programming languages and running on various platforms) in a way similar to the inter-process communication taking place on single computers.
OASIS and the World Wide Web Consortium (W3C) are the steering committees responsible for the architecture and standardization of web services.
Currently available protocols for web services include SOAP (called Simple Object Access Protocol in its first version), the Web Service Description Language (WSDL), and the protocol for Universal Description, Discovery, and Integration (UDDI).
Solutions based on web services apply the service-oriented architecture pattern and are usually characterized by the communication between a service requester (i.e., the system that needs a service), a service provider (i.e., the system that provides the service), and a service broker (i.e., a system that collects and provides information about available services). SOAP is generally used for communication between requester and provider; WSDL can be used to describe services provided by the broker; UDDI can be used to publish available services in service registries.
OASIS is also the steering committee responsible for the standardization of the Business Process Execution Language for Web Services (WSBPEL). The language aims at formally specifying business processes and business interaction protocols. WSBPEL extends the web

services interaction model, enables it to support business transactions, and defines an interoperable integration model to facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

Semantic Web - In its current form, the web presents the user with documents, called web pages, containing links to other documents or information systems. By selecting one of these links, the user can access more information about a particular topic. Since the primarily intended users are humans, the language used to create web pages (i.e., the Hypertext Markup Language, HTML) emphasizes the visual presentation of information and does not provide any support for the semantic classification of information blocks. This approach makes the automated handling of web page content very difficult, if not impossible.

In February 2004, W3C released the Resource Description Framework (RDF) and the Web Ontology Language (OWL) as W3C Recommendations: RDF is used to represent information and exchange knowledge in the web. OWL is used to publish and share sets of terms called ontologies, supporting advanced web search, software agents, and knowledge management. These new technologies allow enhancing usability and usefulness of the web in several ways: web documents become machine-readable through new tags suitable for semantic-based web search; document creators can use common metadata ontologies and maps between vocabularies to annotate their documents so that agents can use the information in the supplied metadata; automated agents can perform tasks for users of the semantic web using this metadata.

Grid Services – Grid Services result from engineering grid computing solutions in accordance with the service-oriented architecture pattern. In this case, resources are virtualized in the form of services.

The Global Grid Forum (GGF) is a community-initiated forum of thousands of individuals from industry and research leading the global standardization effort for grid computing [1]. One of the main results of GGF is the Open Grid Service Architecture (OGSA). OGSA is a service-oriented architecture that specifies a set of distributed computing patterns realized using web services. OGSA can be regarded as the industry blueprint for standards-based grid computing. The Open Grid Services Infrastructure specification (OGSI) is a further result of GGF. OGSI is a set of conventions and extensions of WSDL and its related XML schema to introduce grid-enabled services as a particular type of web services. OGSI introduces the idea of stateful web services and defines approaches for creating, naming, and managing the lifetime of instances of services; for declaring and inspecting service state data; for asynchronous notification of service state change; for representing and managing collections of service instances; and for common handling of service invocation faults.

Since OGSI is not a pure subset of web services and requires a modification of WSDL (Grid WSDL), current tools must be extended to parse and process WSDL for grid services. The Web Services Resource Framework (WSRF) represents a further approach for the specification of grid services. WSRF, like OGSI, defines conventions for managing “states”

so that applications can reliably share changing information. In combination with other web service-standards, the purpose is to make grid resources accessible within a web services architecture. Coupled with WS-Notification, WSRF is a response to, and supersedes, OGSI. WSRF was announced by the Globus Alliance and IBM (with contributions from HP, SAP, Akamai, Tibco and Sonic) in January 2004. The framework is already implemented in available solutions such as version 4.0 of the open source Globus Toolkit for grid computing.
Semantic Web Services - Web services allow a new level of service on top of the current web, since they can be assembled to perform functions or execute business processes. On the other hand, ontologies, which can be considered the basic building blocks of the semantic web, enable the automated handling of documents available on the web based on their content. The combination of the two approaches, which is called semantic web services, opens new possibilities and promises to transform the web from a static collection of information to a distributed device of computation. Technologies related to semantic web services aim at making the web machine-interpretable and will allow the automation of a broad spectrum of activities related to web services, such as discovery, selection, composition, negotiation and contracting, invocation, monitoring of progress, and recovery from failure.
Currently, five initiatives aim at providing semantic mark-up and means for semi-automated discovery, composition and execution of web services: OWL-S (formerly DAML-S) [4], METEOR-S [5], WSMO [6], IRS [14], and ASG .[7].
The OWL-based ontology for services OWL-S is developed by the semantic web services branch of the DAML program.
The METEOR-S project is led by the LSDIS Lab (University of Georgia). The project focuses on adding semantics to WSDL and UDDI, on adding semantics to BPEL4WS, and introduces a semi-automatic approach for annotating web services described using WSDL.
The Digital Enterprise Research Institute (DERI) is the main initiator of the SDK WSMO working group. Major results of this initiative are the Web Services Modeling Ontology (WSMO), the Web Services Modeling Language (WSML), and the Web Services Execution Environment (WSMX). WSMO defines a conceptual model for the semantic description of web services and is based on the Web Service Modeling Framework (WSMF)
The Knowledge Media Institute (KMI) works on the Internet Reasoning Service (IRS), a Semantic Web Services framework, which allows applications to semantically describe and execute web services. The IRS supports the provision of semantic reasoning services within the context of the semantic web. IRS-II and DERI's WSMX are functionally similar, and the newest IRS release (IRS-III) has become a platform and infrastructure for creating WSMO-based semantic web services.
The Adaptive Service Grid Project (ASG) is an Integrated Project supported by the European Union. ASG started on September, 2004, and involves 22 partners from seven countries. The project aims at developing a proof-of-concept prototype of an open platform for adaptive

services discovery, creation, composition, and enactment. Since DERI is one of the involved partners, the ASG and WSMO initiatives share their experience continuously.
Semantic Grid - The combination of technologies applied in the fields of grid services and semantic web services led to the introduction of semantic grid services as semantically and formally annotated, stateful, and grid-enabled web services and to the notion of the semantic grid as a grid computing environment supporting and providing these. According to this, the main goal of semantic grids is to enable the automated processing of data, information, and computing resources within and across virtual organizations. Some of the activities related to this area are coordinated through the Semantic Grid Research Group of the Global Grid Forum.

3 A Practical Example

In this section, we illustrate how the same scenario can be implemented in different ways by applying web services, grid services, or semantic web services.
The example is called “buddy scenario”. The idea behind the scenario is that a user tries to find the location of one of his or her buddies. The user knows the buddy’s name. The user’s request is to find the location of the buddy, preferably in a graphical way. For this example, we assume that the following services are available:

– **S1** - a web service that, for a defined group of buddies, provides the phone number for an identified buddy. This service is based on a local database that provides other services as well, like different personal data for the buddies, such as postal or email addresses.
– **S2** - a location web service provided by mobile phone network providers. This web service takes as input a mobile phone number and provides as result the location of the respective mobile phone. The location data is based on the mobile cell the mobile phone is currently in and is given in this example as GPS data of the center of the mobile cell. We assume that different mobile phone network providers provide such a location web service to locate their users if those agree to this procedure.
– **S3** - a map web service that is able to graphically display GPS data on a map. This web service takes as input GPS data and display a marker at the respective position on a map that displays the area around the location given by the GPS data.

Web Services - In a web service based solution, the user trying to find a buddy is required to find appropriate services. S1 is a local service the user is aware of, so the overall problem of finding the buddy is partially solved: the mobile number is known once this service has been invoked. The user then needs to use web service registries to find services that match the buddy search based on phone numbers. Searching a service registry for services that provide a location for a given mobile phone number will result in the services provided by the different mobile phone network providers. The user then needs to select the provider of the buddy either by trying the different services or by knowing the buddy’s provider based on the

number. Once the location web service is identified, the user knows that the result is given as GPS data. With this information, finally, the map service S3 can be identified and invoked. Thus, the user can now invoke the selected services manually, passing the result of one service to the next service, until, finally, the map with the buddy's location is displayed.
In the web service based solution, the user needs to autonomously search for appropriate services, understand and match their interfaces, and finally, invoke the services in the proper order forwarding the results from one service as input to the next.
Grid Services - From a user's point of view, the situation is not very different in a grid service based solution, since the grid is transparent to its users. The difference can be that the different services are available on the same grid and the search for appropriate services is thus simplified. There is, however, a big difference on the service provider side. They can exploit the sharing of grid resources to provide a higher quality of service, resulting in services that are, for instance, faster, more available, and more reliable.
Semantic Web Services - For the user, the situation changes dramatically if the solution is based on semantic web services. This is because the user can directly invoke a request that finds and displays the buddy's location providing as input the buddy's name only. The semantic annotation of the web services enables the automatic selection of the phone number service S1, the location service S2, and the map service S3, as well as their composition based on the types of input and output parameters of the involved services, as well as on their pre and post conditions. The automatic invocation forwards the results from one service as input to the next service and provides the final result to the user. The selection of the right mobile provider can be either done by trial and error or by automatically discovering the appropriate service by relating the semantic annotations given by the user with ontological information contained with in the service that associates phone numbers with mobile phone providers.
The difference for the users of semantic web services is that they are required to augment their service request with semantic information. If, however, they provide this information, they do not need to search for appropriate services, find services that fit and together solve their problem, and invoke the services, since these activities can all be performed automatically, based on the provided semantic information.

4 Engineering Processes

In this section, we present the results of a literature survey we performed in order to determine available reference processes for engineering solutions in the semantic grid domain. Processes for engineering and setting up grid infrastructures are out of the scope of the survey. Keyword searches were performed in databases such as TEMA and Fraunhofer Publica. Other sources of information were the Internet (in particular, the domains related to DAML, DERI, GGF, Globus Alliance, IBM, and W3C) and the publications of thematically

related international conferences (e.g., Grid Services Engineering and Management GSEM 2004 [23]).
In the following, processes are arranged in three main groups according to their application field: web services, grid services, and semantic web services.
Web Services - Zimmermann et al. [11] discuss the Service-oriented Analysis and Design (SOAD) approach, which aims at helping organizations to discover new business opportunities and threats. Solutions engineered following this approach should be based on reusable services, which in turn must use and provide well defined, standard-compliant interfaces. In the following, a brief description is provided.
Describe the business scenario - The objective of this activity is to create a workflow description that will be used for modeling the service. There are no suggested notations or techniques that could be mapped to this activity.
Create a conceptual service model - Traditional requirements analysis techniques can be performed through interviews or group meetings with stakeholders in order to discover candidate services. Another possibility is to follow the Component Business Modeling (CBM) technique. CBM is a technique that could help in deriving services in a top-down manner [37]. It provides a framework for viewing the business as a network of discrete services, turning the services into unique building blocks.
Create a business process model - Business process models shall be described after having identified the candidate services. They shall be described as a sequence of operations/services performed with a specific business goal in mind. The creation of a business process model consists of two steps: *create a service states model* and *create a business interaction model*. Once the business process model has been identified, techniques from enterprise architecture frameworks and object-oriented analysis and design can be used for implementing and deploying the service(s).
Similar approaches and examples for identifying, designing, implementing, and deploying services were found in the telecommunications domain [8],[9]. Here, one proposal of a process model and its application to provide a means for systematically creating services with interoperability capabilities is provided in [10]. In this approach Service Information Objects (SIO) and their relationships are drawn in a conceptual model. Then, a first sketch of the services interaction is recorded in a behavioral model. Service Information Objects identified in the previous phase are structured into Computational Objects (COs). This is done by encapsulating service information objects into computational objects with an operational interface as a wrapper. Detailed representations from the conceptual model and the behavioral model are recorded in interactions and class diagrams respectively.
Grid Services - Many examples can be found on Internet about how to develop user grid services using Globus Toolkit [16], [17], [24]. A generalization of the process is described in the following.

Create the service interfaces - Grid services interfaces can be defined either manually by using WSDL or automatically.
Generate stub and support code - After defining the grid service interface, stubs can be created, which, in turn, enable the service to be accessed through protocols such as SOAP over HTTP.
Write server-side-implementation code - Once stubs have been generated, the grid service can be implemented following the guidelines provided in [19].
Write the client-side implementation code - In order to create a client that accesses the grid service, an implementation of the client side is needed. Possibilities can be to implement a web interface (e.g., a portal) or to implement a language specific API.
Deploy and test the grid service - In order to deploy the service, the developer must write an XML descriptor file, which describes and configures the service. Finally, the file can be included in any suitable grid services hosting environment.
Examples were also found based on a different platform: the OGSI.net tool. In this case, the activities *create service interface*, and *generate stub and support code* are not performed; instead, after implementing the logic of the service, annotations are attached to the service implementation with the goal of exposing the operations, data, and policies defined in the service as metadata. The annotations can be processed by static analysis tools and the container's runtime system to transform the author's service logic into a grid service without requiring the author to have a detailed knowledge of the container [18].
Semantic Web Services - As introduced in section 2, an infrastructure allowing the semantic annotation of services is needed by software agents to handle services autonomously, i.e., to reason about services. Section 2 also introduces the, currently, five major initiatives addressing the semantic-oriented handling of services: OWL-S [4], METEOR-S [5], WSMO [6], IRS [14], and ASG [7].
Cabral et al. [13] provide an interesting overview and comparison of the approaches IRS-II, OWL-S, and WSMF/WSMO. The article states that none of these approaches provides a complete solution in this area. The approaches show complementary strengths. Particularly, the task of selecting services available in registries still requires intervention by human users. The article also highlights the similarity of the objectives in the face of a great diversity in reasoning support based on different logic and ontology frameworks. Table 1 shows the mapping of main concepts from the three approaches as presented in [13] enhanced with the concepts from METEOR-S and ASG, which were not considered in the original article. According to the article, three orthogonal characterization dimensions are distinguished, which relate to the requirements for semantic web services at the business, physical, and conceptual levels. The *usage activities* dimension defines a set of functional requirements for a framework of semantic web services (i.e., automated publishing, discovery, selection, composition, invocation, and deployment of services). The *architecture* dimension proposes a list of needed components (i.e., register, decomposer, reasoner, invoker, and matchmaker).

The *service ontology* dimension represents a minimal set of concepts needed for the semantic-rich description of a service (i.e., input, output, pre-condition, post-condition, cost, atomic service, composite-service, and category). Already available tools are also listed.

Table 1. Semantic Web Services comparison

Dimension	IRS-II	OWL-S	WSMF	METEOR-S	ASG
Usage Activities	Publishing, Selection, (Task Achievement)	Composition, Discovery, Invocation	Discovery	Publication, Discovery, Composition, Abstract Process Creation	Composition, Enactment, Fault-tolerant service execution, Re-negotiation, Re-composition, Service level agreement, Negotiation, Monitoring and profiling
Architecture	Server, Publisher, Client	DAML-S Virtual Machine, Matchmaker	Service Registry, Profile Crawler	Abstract Process Designer, Semantic Publication and Discovery Engine, Constraint Analyzer, Execution Environment	Ontology and Service Specification, Discovery Database, Service Composer, Service Creation, Mediated Replanning, Negotiation Manager, SLA Management, Service Profiling, Workflow Enactment, Deployment Service, Invocation Service
Service Ontology	Task / PSM Ontology	OWL-S	WSMO	WSDL-S	ASG Ontology
Application Tools	IRS Browser and Editor; Publisher; Java API	WSDL2DAML-S	Query interface	Semantic Web Service Developer / Annotator	Prototypical implementations of the subsystems mentioned under Architecture

Terziyan et al. [12] present another analysis of the state of the art in this field. The article claims once again the need for extensible ontology frameworks for bringing web services to their full potential due to the semantic handling of services. According to [13], also [12] argues that semantic web technologies are neither mature nor state of practice in the industry. Table 1, for example, makes clear that even the formalisms applied for describing services in a semantic-rich way, i.e., the service ontology, differs in all approaches presented.
As a consequence of the lack of mature solutions based on accepted standards, there is no evidence for standardized processes aimed at engineering solutions based on semantic-rich descriptions of services. Commented engineered solutions can be found in the literature (for an example of semantic web services applied in a wireless environment, see [15]; for another example concerned with automated selection of web services refer to [27]), from a process-related point of view such examples deal with ad hoc, prototypical implementations performed without the aid of a surrounding reference process. The ASG project address this lack of methodology explicitly and a first attempt to analyze and formalize the engineering processes in this field can be found in [28].

5 Conclusions

According to [2], there is a wide spectrum of benefits to be gained from grid computing, but there exists no “silver bullet” for achieving them. Each organization must build its own solution, and special care must be taken if the final purpose is to exploit such benefits. One possible strategy for establishing the foundations of a semantic grid solution can be to identify similar resources from a business perspective, handle them in a semantic oriented way, and then to virtualize them. In this context, virtualization means making information available whenever, wherever it is needed [3], [26]. This way, more concrete steps can be performed towards an intra- or inter-organizational grid solution.
Approaches that can be of help for identifying similar resources from a business perspective can be found in the service-oriented domain as well as in related domains such as telecommunications [8], [9], [10], [11]. These domains do not show substantial differences in their processes when transforming a business idea into a service model. Both suggest capturing the business idea through scenarios, then creating a conceptual service model that reflects service concepts involved in the mentioned scenarios, followed by a refinement of the service flow by defining operations, relationships to external services, and states of the service, and finally, orchestrating the service by defining rules and interaction models.
Additionally, the actual tendency of major software vendors is to allow choreography of business processes by integrating web services and process engines [20], [21], [25] (e.g., Websphere Integration Server Foundation, Business Works, Oracle BPEL Process Manager). This is realized on top of their web application servers, which in turn are being developed compliant to OGSI (e.g., the IBM WebSphere Application Server - Express V5.0.2).
Concerning the engineering of solutions aimed at handling services in a semantic-oriented way, five different approaches have been proposed: IRS, OWL-S, WSMF, METEOR-S, and ASG. Although they address similar objectives, they also turn out to be different in terms of reasoning support, mainly due to different underlying logic and ontology frameworks. The approaches show complementary strengths and the importance of standardization was already recognized. Cooperation between IRS and WSMO, on the one hand, and WSMO and ASG, on the other, shows that the approaches are trying to converge. Currently, none of the proposed frameworks can be seen as a reference capable of supporting standard processes for the semantic annotation of services and the engineering of intelligent machines able to autonomously apply and combine the annotated services for coming up with a solution to a given problem. As a first consequence, the engineering processes applied in this area are still ad hoc and unstable and their analysis and formalization is just beginning. As a further consequence, since the engineered prototypes mostly serve as a proof of concept, it is difficult to distinguish between the processes aimed at engineering such intelligent machines and those processes aimed at engineering the needed (and currently only partially available) underlying infrastructure.

Standards for service-oriented architectures are leading, in the last few years, to standard processes for engineering solutions based on Web services. In a similar way, we expect that shared formalisms for semantic-rich service annotation, once established, will allow the standardization of basic technologies, which is the basis for a comprehensive methodology aimed at engineering semantic grid solutions systematically and rapidly.

Acknowledgments

This work has been funded by the European Commission in the context of the integrated project Adaptive Service Grid (ASG) (FPS-IST 004617). We would also like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) and Harald Mayer (Hasso-Plattner Institute at the University of Potsdam) from the ASG consortium for reviewing an early version of the article. The buddy scenario we used in section 3 was developed in the ASG consortium as an example for discussing principles, techniques, methods, and tools among the project partners. We would like to thank all project members who contributed to the buddy scenario, especially Dominik Kuropka (Hasso-Plattner Institute at the University of Potsdam) for providing this example.

References

[1] Global Grid Forum: OGSA FAQ Sheet (http://www.gridforum.org/L_WG/News/OGSAFAQ_Handout.pdf) viewed 15 Feb 2005.
[2] Harris, C. (2004): Getting started with Grid. IBM Global Services, 2004.
[3] Palfreyman, J. (2004): Grid Explained. IBM Global Services, 2004.
[4] OWL Services Coalition (2003): OWL-S: Semantic Markup for Web Services, (http://www.daml.org/services/owl-s/1.0/), viewed 15 Feb 2005.
[5] A. A. Patil, S. A. Oundhakar, A. P. Sheth, K. Verma: Semantic Web Services: Meteor-S Web Service annotation framework, Proceedings of the 13th WWW conference, May 2004.
[6] D. Roman, H. Lausen, and U. Keller: Web Services Modeling Ontology Standard, WSMO Working Draft v02, 2004.
[7] Adaptive Service Grid (ASG) Project: Project Home Page (https://asg-platform.org/cgi-bin/twiki/viewauth/Internal/WebHome), last visited February 23, 2006
[8] D.X. Adamopolous, G. Haramis, C.A. Papandreou: Rapid prototyping of new telecommunications services: a procedural approach, Computer Communications 21, pp. 211-219, 1998.
[9] D.X. Adamopoulus, C.A. Papandreou: An integrated object-oriented approach to telecommunications service engineering, Proceedings of IFAC/IFOR/IMACS/IFIP LSS '98, Rio, Greece, pp. 834-839, 1998.
[10] D.X. Adamopoulus, G. Pavlou, C.A. Papandreou: An integrated an systematic approach for the development of telematic services in heterogeneous distributed platforms, Computer Communications 24, pp. 394-415, 2001.
[11] O. Zimmermann, P. Krogdahl, C. Gee: Elements of Service-oriented Analysis and Design: An interdisciplinary approach for SOA projects, Available at http://www-106.ibm.com/developerworks/Webservices/library/ws-soad1/
[12] V. Terziyan, O. Kononenko: Semantic Web Enabled Web Services: State-of-Art and Industrial Challenges. ICWS-Europe 2003, M. Jeckle and L.-J. Zhang (Eds.), LNCS 2853, pp. 183–197, Springer-Verlag Berlin Heidelberg 2003

[13] L. Cabral, J. Dominguez, E. Motta, T. Payne, F. Hakimpour: Approaches to Semantic Web Services: An Overview and Comparison. First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece, May 10-12, 2004, Proceedings, Available at http://kmi.open.ac.uk/projects/irs/cabralESWS04.pdf
[14] E. Motta, J. Dominguez, L. Cabral., M. Gaspari: IRS-II: A Framework and Infrastructure for Semantic Web Services. In: Fensel, D., Sycara, K., Mylopoulos, J. (volume eds.): The SemanticWeb - ISWC 2003. Lecture Notes in Computer Science, Vol. 2870. Springer-Verlag, Heidelberg (2003) 306–318
[15] F. Gandon, N. Sadeh: Semantic Web Technologies to Reconcile Privacy and Context Awareness, Web Semantics Journal. Vol. 1, No. 3, 2004.
[16] A.E. Walsh: Building computational grids using the Globus Toolkit, Dr. Dobb's Journal. September 2003.
[17] L.J. Zhang, Q. Zhou, J.Y. Chung: Introduction to a Grid Architecture and Toolkit for Building Grid Solutions, Available at: http://www-106.ibm.com/developerworks/Grid/library/gr-grid2/
[18] G. Wasson, M. Humphrey: Attribute-Based Programming for Grid Services, Computer Science Department University of Virginia, 2004, Available at http://www.cs.virginia.edu/~gsw2c/OGSIdotNet/Grid_service_programming.GGF9.pdf
[19] The Globus Alliance: Java Programmer's Guide Core Framework, 2003, Available at: http://www-unix.globus.org/toolkit/3.0/ogsa/docs/java_programmers_guide.html.
[20] M. Reapple: IT-Ballet. Vier Process Engines im Vergleich (Comparison of four process engines), iX- Magazin für Profesionelle Informationstechnik. 2004.
[21] W. Liu, G. Goldzmidt, J. Joseph: On demand business process life cycle, Part 5: Workflow development, deployment, and testing, Available at: http://www-128.ibm.com/developerworks/library/ws-odbp5/?ca=dnt-64
[22] CERN: Grid-like projects http://gridcafe.web.cern.ch/gridcafe/Gridhistory/gridlike.html viewed 20 April 2005.
[23] M. Jeckle, R. Kowalczyk, P. Braun (Eds.): Grid Services Engineering and Management. Proceedings of the First International Conference GSEM 2004, Erfurt, Germany, Springer-Verlag, 2004.
[24] L. Ferreira, A. Thakore, M. Brown, F. Lucchese, H. RuoBo, L. Lin, P. Manesco, J. Mausolf, N. Momtaheni, K. Subbian, O. Hernandez: Grid Services Programming and Application Enablement. Available at: http://www.redbooks.ibm.com/.
[25] M. Keen, J. Cavell, S. Hill, C. K. Kee, W. Neave, B. Rumph, H. Tran: BPEL4WS Business Processes with WebSphere Business Integration: Understanding, Modeling, Migrating. Available at: : http://www.redbooks.ibm.com/
[26] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke: Grid Services for Distributed System Integration. IEEE Computer. June 2002 (Vol. 35, No. 6) pp. 37-46.
[27] Daniel Mandell, Sheila McIlraith: Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. Proceedings of the 2nd International Semantic Web Conference (ISWC2003), Springer Verlag, 2003
[28] Laures, G.; Meyer, H.; Breest, M.: An Engineering Method for Semantic Service Applications, in Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05), ICSOC, Amsterdam, The Netherlands, December 2005.

Extended Service Binder: Dynamic Service Availability Management in Ambient Intelligence

André Bottaro*, Anne Gérodolle*
*France Telecom R&D

Abstract. *This paper explores Service Oriented Architecture in Ambient Intelligence with focus on distribution and dynamic selection. We aim at building a framework transparently handling these aspects.*

Keywords: Service Oriented Architecture, Ambient Intelligence, OSGi.

1 Introduction

Ambient Intelligence vision assumes that the computers will fade into the background. Building consistent context-aware application inside and outside the home with heterogeneous devices which spontaneously enter and quit the network is the objective of software designers in the field.

This paper proposes a simple declarative model to automate the mutual discovery and binding of components running on distributed network nodes. The model and the associated software architecture above OSGi [14] and *Service Binder* [3] are described in details. Service Binder Model is now integrated in OSGi R4 specification as *Declarative Services* [13]. A middleware hiding the multiplicity of service discovery and distant communication protocols can be built above this model. Many projects aim at defining such a pervasive middleware [5], [4], [15]. Current European IST projects like Amigo, Daidalos and SeCSE tackle with service composition in distributed networks. This paper introduces a pluggable architecture addressing a generic way to interact with multiple service discovery and distant communication protocols.

The implementation specification lies on several technological choices:

- The core framework is Java-based and uses the OSGi standard R3 specification [14] and Service Binder Model [3].
- Discovery technologies are well-known standards: UPNP/SSDP [18], Jini [10], SLP [7], Web Services [6], and CORBA [12].

Service Oriented approach supporting the needs of targeted pervasive applications. is described in Part 2. The implementation of the concepts is detailed in Part 3. Part 4 describes a scenario of use. A conclusion is given in Part 5.

2 A Service Oriented Framework

In smart environments, devices may randomly enter and quit the network., users come and quit the environment with PDAs and mobile devices, install new devices, old devices are updated or replaced, etc. Moreover, devices may deliver services with varying QoS and properties. Some devices may become useless while others may become relevant in distinct situations. Simply identifying every device in our application and being able to deal with the dynamic availability and relevance of these devices are important needs in pervasive computing.

Service oriented programming is an adequate paradigm dealing with these needs. In Service Oriented Architecture, software components register services with associated properties to be requested by other components acting like service clients. Registration and requests are based on the use of a service repository which can possibly be centralized, replicated or distributed through the use of multicast mechanisms.

Distributed Service Oriented Architecture. Numerous Service Oriented Architecture specifications address the issue of Service Discovery and the issue of Distant Communication in distributed environments. If most of the specifications address the same generic issues, they have distinct objectives and are made efficient for distinct environments [15]. UPnP specifies the use of several web-oriented protocols in order to build an attractive solution for small and dynamic networks. Jini aims at specifying a Java middleware for distributed applications on larger networks. SLP is also meant for SOHO (Small Office, Home office) networks. The specification proposes an interesting way to move from multicast discovery mechanisms to centralized one. Web Services may target SOHO network with specifications like WS-Discovery (Web Services Dynamic Discovery). Several specifications make service discovery relevant on CORBA: Naming Service, Trading Service, Event Service. CORBA specification is really large and is not meant for a particular environment. Other comparison details are found in [2].

However, those middlewares do not hide dynamic service availability to application developers. The latter have to repeatedly write error-prone code to request available services and listen to service arrival and departure. Distribution is a non-functional need which has to be dealt by the underlying framework.

Dynamic service availability is shown as a non-functional need in [3][8] which describe a model named Service Binder automating service discovery and service binding in a dynamic service environment. Service Binder model [16] is a simple service oriented model above OSGi Release 3 [14] enabling developers to encapsulate consistent code into components providing and requiring services to each other. The provided and required services are simply declared by every component. A centralized instance manager factory builds an instance manager for every instance of declared components. This instance manager manages component life cycle according to the availability of required services. The implementation of this model greatly simplifies application programming. The developer should only think about the granularity of his application in terms of components and specify what generic services could be provided by components for internal or external application purposes.

However, Service Binder model shows some limitations in order to be the underlying framework of our architecture. First, service properties declared by every service provider are statically written in an XML file and could not be dynamically changed at runtime. This limitation is explained below. Second, the automation of service discovery and service binding is only performed on a single platform. These limitations are overcome by the extensions proposed in part 3.

3 Hiding distribution and managing dynamic service properties

3.1 Hiding distribution mechanisms complexity.

The extension of service discovery and distant communication to distributed mechanisms has to be transparently managed by the underlying framework. This section describes how mechanisms are extended over Service Binder Model in the proposed architecture. This extension is called Extended Service Binder.

Extended Service Binder relies on “export and binding factories” to build and use remote object references, and on “lookup services” to register references with properties and find remote references on the network. Bundles and services using Service Binder extended mechanisms interact with all available OSGi services on the local platform and exported ones on the network. Extended Service Binder is a container managing non-functional aspects.

Remote binding relies on the export-binding pattern defined in ODP [9] and defines two Java interfaces defining the concepts of export and binding factories. An “export factory” is a service that makes a Java object remotely available. The result of “Exporting a service” is a “binding description” that can be serialized and published using a discovery protocol.

Symmetrically, a “binding factory” is used on the client side to bind to a given service, given a “binding description”. A binding factory provides a “bind” method that takes a binding description as parameter and returns an object called “proxy” or “stub”. This stub can then be used by the client to communicate with the remote object.

Extended Service Binder relies on Lookup Services representing remote or local non-OSGi service registry on the platform. They implement the same service interface enabling service providers to register and deregister services, service clients to look up services and be notified by events concerning filtered services.

Extended Service Binder transparently manages distribution. A simple boolean value in a metadata file can be set by the developer to indicate that its service providers are wished to be announced on the network with one or every available discovery protocol ("registry='local, slp,upnp' or "“registry='*'” to be seen in examples, part 4.3) or that its service clients may bind remote providers (“local-only=false” in service client XML description, part 4.3). Transparency is reached with static or dynamic stub generation. A proxy for every targeted technology has to be generated for any used remote service provider. Extended Service Binder structure is illustrated in Fig. 1.

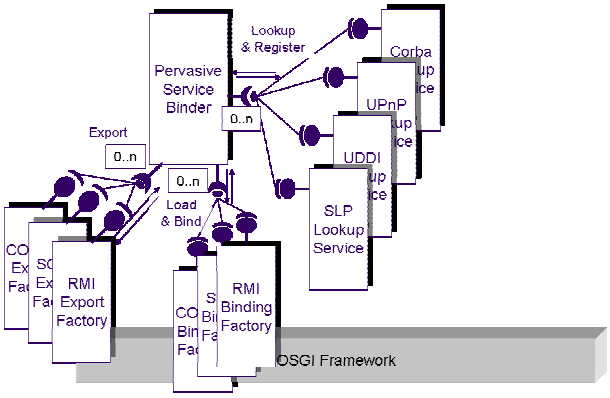


Fig. 1. Extended Service Binder

3.2 Extend the component XML metadata model to manage dynamic properties.

The service dependency management is a non-functional aspect which has to be held separately. Using an external XML file is a natural way to make the separation clearer.

On the one hand, a service requirement in Service Binder Model is described in an XML reference to a service interface with binding and unbinding methods and some dependency properties: policy, cardinality, filters (see example, part 4.3). In order to dynamically add selection mechanisms after service filter at runtime, the declaration of a dynamic selection method could be added. It will be called at service dependency instantiation time. A method called "sort-method" is to be seen in the example, section 4.3.

On the other hand, a provided service in Service Binder Model is described in an XML reference to an object class implementing the provided service interface with statically defined properties (see example, part 4.3). In order to dynamically modify the properties of a provided service, similar extensions are to be held. The declaration of a method setting dynamic properties in provided service declaration and a simple method calling Service Binder to modify service registration through Component Context object could be added. A method called "property-method" is to be seen in the example, section 4.3.

4 An example of pervasive application: audio streaming follow-me

In “follow-me” applications [11] , a service follows the user as he/she moves by borrowing interfaces from devices in the user’s vicinity. Following this concept, Extended Service Binder has been applied to build an “audio streaming follow-me” service. For the sake of simplicity, only a subset of the involved components is detailed here.

4.1 The service offered to the user

When the user moves inside the home, the audio stream "follows" him/her by using the best available rendering resource in his/her vicinity, as illustrated in **Fig. 2**.

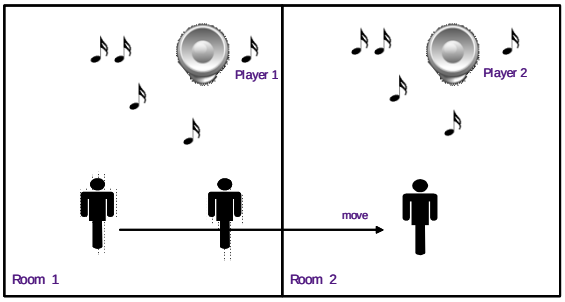


Fig. 2. Follow–Me: When the user moves from a room to another, the player of the second room starts playing and the first stops playing after a while.

4.2 Architecture

To offer this service, the following components were identified. The links between these components are illustrated in **Fig. 3**. The first four components provide services whose Java interfaces are written below.

- A user localisation service based on several localization techniques.
- Several audio renderers able to connect and render an available audio stream.
- One or several yellow pages servers able to give a list of available stations
- A "control point" and "control point factory" whose role is detailed later on.
- A remote control enabling the user to select an audio content and control rendering.

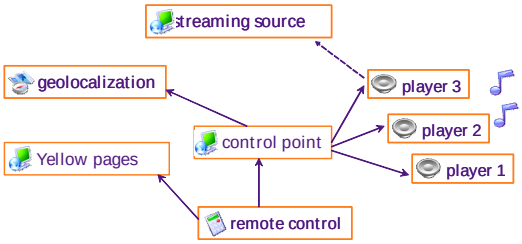


Fig. 3. Architecture of the follow-me application.

4.3 Bundles, components and algorithm details

Each of the main components is packaged in an OSGi bundle and is described in the Extended Service Binder model.

The component providing the AudioRenderer interface is hosted by a bundle which must be deployed on every device with loudspeakers that may be used for audio rendering.

```
<bundle>
  <component class="renderer.AudiorendererImpl">
<onregister property-method="getDeviceProperty"/>
<provides service="api.AudioRenderer" registry="*"/>
  </component>
</bundle>
```

Before registering the AudioRenderer component, the Extended Service Binder calls the "getDeviceProperty" method. This method gives the location of the speakers and the "quality" (built-in loudspeakers, external loudspeakers, hi-fi, etc.) of the rendering. The service is then registered with these properties.

The control point bundle provides a factory of "control points". A specific control point is dedicated to any active user and ensures that the stream chosen by the user is rendered on the adequate platform. To that purpose, it needs to access the localisation service and the available audio renderers. The control point bundle is typically deployed on an "always on, always connected" OSGi platform.

```
<bundle>
<component class="control.CPFactoryImpl">
  <provides service="api.CPFactory" registry="*"/>
  <instantiates class="pack3.ControlPointImpl">
   <provides service="api.ControlPoint" registry="*"/>
   <requires
        Service=″api.LocalisationService″
        cardinality=″1..1″ policy="dynamic" local-only="false"
        bind-method="bindLocService" unbind-method="unbindLocService"/>
   <requires
        Service=″api.AudioRenderer″ local-only="false"
        cardinality=″0..1″ sort-method="sortRenderers"
        bind-method="bindRenderer" unbind-method="unbindRenderer"/>
  </instantiates>
</component>
</bundle>
```

The control point currently keeps: a current renderer, a former renderer, and a current context (i.e. the multimedia resource being currently played, the volume information and other rendered properties).

The "sortRenderers" method chooses the available renderer which is in the same room as the users, and (if several) the one with the best rendering quality. This method is called by the Service Binder when the current renderer quits the network, when a new audio renderer enters the network, when an audio renderer modifies its registration, or when discovery is asked to be refreshed by the control point. This refreshment can be asked when the location of the user changes. According to the result of sortRenderers, the method "bindRenderer" may be called by the Service Binder if another audio renderer service must be bound. Whenever the user moves from a room to another, the Extended Service Binder binds the control point to the best available audio renderer.

The "bindRenderer" method sets the local "current renderer" and "former renderer" information and initializes the current renderer with the current playing context. This method call is followed, after

some delay, by the call to "unbindRenderer" method calling the "stop" method of the former renderer. Thanks to the delay between the calls to bindRenderer and unbindRenderer, the user can still listen to the audio stream while the new renderer starts buffering the audio stream.

5 Conclusion and future work

We have presented a middleware architecture called Extended Service Binder, which facilitates the building of pervasive applications. The following features are delivered: transparent access to remote services available through any service discovery protocol, selection between services according to the current context.

Thanks to this architecture, a real peer-to-peer dynamic composition occurs at runtime. Service composition is spontaneously held by the different components on the distributed platforms. Service binding is optimized: Two components running on a single platform are directly bound in the local virtual machine with direct method calls whereas a protocol for distant communication is automatically used if they run on distinct platforms.

The drawback of this approach is that Java interfaces must be agreed in advance between service providers and requesters. Future work will tackle the interoperability between similar interfaces defined through distinct syntactic languages. The definition of useful ontology is currently ongoing with the use of OWL-S. The semantic Web services approach [1] leverages the use of formal and machine-understandable descriptions to enable a flexible matching between heterogeneous services. We are currently investigating such an approach in a pervasive computing context.

6 Acknowledgments

We are particularly grateful to the RNRT (French Research Network in Telecommunications) through the PISE project, which offers us the opportunity of trying out some of our ideas presented in this article in an industrial environment.

We would like to also thank IST, the European research program. This work was partially founded by IST through the Amigo project in Ambient Intelligence.

7 References

1 Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, Katia Sycara, "DAML-S: Web Service Description for the Semantic Web", First International Semantic Web Conference, Sardinia, Italy, June 2002

2 Christian Bettstetter, Christoph Renner, “A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol”, In Proc. EUNICE Open European Summer School, Twente, Netherlands, Sept 13-15, 2000

3 Humberto Cervantes, Richard S. Hall, “Automating Service Dependency Management in a Service-Oriented Component Model”, Proc. 6th Wksp, Component Based Engineering, May 2003

4 Caroline Funk, Christoph Kuhmünch, Christoph Niedermeier, "A Model of Pervasive Services for Service Composition", CAMS05: OTM 2005 Workshop on Context-Aware Mobile Systems, Agia Napa, Cyprus, October 2005

5 Paul Grace, Gordon S. Blair, Sam Samuel."ReMMoC, "A Reflective Middleware to Support Mobile Client Interoperability", Proceedings of International Symposium on Distributed Objects and Applications (DOA), Catania, Sicily, Italy, November 2003

6 Steve Graham, Doug Davis, Simeon Simeonov, Glen Daniels, Peter Brittenham, Yuichi Nakamura, Paul Fremantle, Dieter Koenig, Claudia Zentner, “Building Web Services with Java”, Sams Publishing, Second Edition, 2004

7 Erik Guttman, Charles Perkins, John Veizades, Michael Day, “Service Location Protocol, Version 2”, RFC 2608, June 1999
8 Richard S. Hall, Humberto Cervantes, “Challenges in Building Service-Oriented Applications for OSGi”, IEEE Communications Magazine, May 2004
9 ITU-T & ISO/IEC, “ODP Reference Model: Overview, Foundations, Architecture”, Recommendations X.901, X902, X903 & International Standards 10746-1, 10746-2, 10746-3, 1995
10 Jini.org, “The Community Resource for Jini Technology”, http://www.jini.org
11 Robin Kirk and Jan Newmarch, "A Location-aware, Service-based Audio System", IEEE Consumer Communications and Networking Conference, 2005
12 The Object Management Group, “Common Object Services Specification”, http://www.omg.org/technology/documents/formal/corbaservices.htm
13 OSGi Alliance, OSGi R4 Core Specification, October 2005
14 OSGi Alliance, “OSGi Service Platform – Release 3”, March 2003
15 Pierre-Guillaume Raverdy, Valérie Issarny, “Context-aware Service Discovery in Heterogeneous Networks”, Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2005), June 2005
16 Humberto Cervantes, Richard S. Hall, “Simplifying application development on OSGi”, documentated open source code, http://gravity.sourceforge.net/servicebinder/
17 Sun Microsystems, “Java RMI”, http://java.sun.com/products/jdk/rmi/
18 UPnP.org, “The UPnP Forum”, http://www.upnp.org

Service-Oriented Development In a Unified framework (SODIUM) – Future Research Challenges

A.-J. Berre[1], H. Hoff[1], David Skogan[1], A. Tsalgatidou [2], G. Athanasopoulos[2], M. Pantazoglou[2]

[1] SINTEF Information and Communication Technology
P.O.Box 124 Blindern, N-0314 Oslo, Norway
{arne.j.berre, hjordis.hoff, david.skogan}@sintef.no
[2] National & Kapodistrian University of Athens (NKUA),
Department of Informatics & Telecommunications, Athens 15784, Greece
{atsalga, gathanas, michaelp} @di.uoa.gr

Abstract. Technology trends in software development signify a move from component-based to service-oriented development. E-Services are the building blocks of service-oriented applications and are mainly instantiated by web, grid and p2p services. The development of service-oriented applications requires support for service discovery, composition and execution. However, the discovery and composition of appropriate services is not an easy task, due to the heterogeneity and incompatibility between the architectural models, protocols, and standards employed by web, grid and p2p services for description, discovery and composition. Therefore, there is a need for appropriate support to discover and integrate heterogeneous services. In this paper we present future research challenges, based on the results and experiences from the EU IST project SODIUM. SODIUM comprises a generic service model called GeSMO, a set of languages and tools based on GeSMO, as well as related middleware and supporting methodology for the discovery and composition of heterogeneous services in a unified way.

1 Introduction

Current trends in software engineering signify a move from component-based to service-oriented development (SOD). SOD promises the development of interoperable, loosely-coupled, distributed service-oriented applications. E-Services are the building blocks of service-oriented applications and are mainly instantiated by web, grid and p2p services. Thus, new service-oriented applications, rather than being built from scratch, are being developed as service compositions by exploiting the large number of available services. However, the discovery and composition of appropriate services is not an easy task, due to the heterogeneity and incompatibility between the architectural models, protocols, and standards employed by web, grid and p2p services for description, discovery and composition. This paper briefly describes an approach called SODIUM (Service-Oriented Development In a Unified framework) which attempts to bridge this gap by offering a collection of models, languages and open source corresponding middleware to support the discovery and

composition of heterogeneous (web, p2p, and grid) services**,** in an open and unified manner.

The use of the various SODIUM components is envisaged as follows. When a developer starts the design of a service composition, s/he first needs to specify the various composition tasks and the control flow between them. These tasks can be executed by various types of services (rather than being programmed from scratch). These services may not be known initially. Therefore, by following a top-down approach for developing a service composition, there is a need to model requirements for appropriate services which satisfy specific composition tasks. SODIUM provides a Visual Composition Language (VSCL) and an associated VSCL Editor (described in section 2.2) which support this modeling task. The next step is to search for appropriate services which can satisfy the requirements of each task in the service composition. There exist a large number of heterogeneous web, p2p and grid services with incompatible protocols and standards which makes their discovery a cumbersome task. SODIUM provides a Unified Service Query Language (USQL) and an associated query engine (described in section 2.3) that support the discovery of heterogeneous services in a unified way. Both semantic and Quality-of-Service (QoS) information are utilized to improve the discovery. Selected services substitute the requirements in each composition task resulting in a concrete service composition model. Next, VSCL graphs are mapped to USCL (Unified Service Composition Language) descriptions which are executed by the SODIUM execution engine, described in section 2.4. The main purpose of the latter is to provide an efficient, reliable and scalable platform for executing compositions of heterogeneous services.

In order to support the discovery and composition of heterogeneous services, SODIUM provides a Generic Service Model (GeSMO), described in section 2.1, which constitutes the common basis for the development of SODIUM languages and tools.

In the next section (section 2), we present the overall architecture of the SODIUM platform and its constituent components. Section 3 presents future research challenges with respect to each of the SODIUM platform tool and finally Section 4 illustrates our concluding remarks.

2 The SODIUM Approach

This section provides an overview of the SODIUM platform. As we can see in **Fig. 1**, SODIUM introduces a set of languages, tools and associated middleware as well as a conceptual model - called GeSMO (which provides the basis for all SODIUM languages) and a methodology (which provides guidance to a developer in the composition process). The languages introduced by the SODIUM platform are:

- A *Visual Service Composition Language (VSCL)* for designing service compositions at multiple levels of details.
- A *Unified Service Composition Language (USCL)* to facilitate the construction of executable compositions of heterogeneous services.
- A *Unified Service Query Language (USQL)* to cater for the open and unified discovery of heterogeneous services enabling the preservation of the autonomy of service registries.

With respect to tools and middleware, the SODIUM platform provides the following:

- A Visual Service Composition Suite comprising:
  - A *Visual Editor* enabling the construction and analysis of VSCL Graphs.
  - A Translation mechanism enabling the transformation of the VSCL graphs into USCL.
- A Run Time Environment comprising components necessary for the execution of the composite services:
  - A search engine, namely *USQL Engine*, which submits queries to heterogeneous service registries, utilizing USQL.
  - A *Workflow Execution Engine* which executes workflows written in USCL. The workflow engine invokes the different types of services and/or submits USQL queries to the USQL Engine and subsequently invokes the returned services.

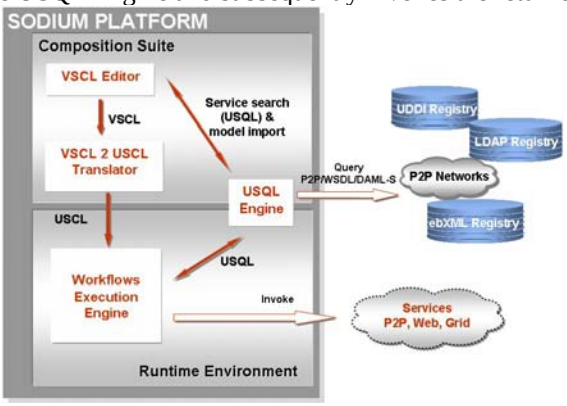


Fig. 1. SODIUM Platform overview architecture

In the following, we describe the Generic Service Model (GeSMO) and the three main components of the SODIUM platform along with the respective languages, i.e. the Visual Editor, the USQL Engine and the Execution Engine.

2.1 Generic Service Model

An important asset of SODIUM is the Generic Service Model (GeSMO) which provides: 1) a set of concepts common in all service types and 2) extensions to these common concepts which describe the distinct characteristics of the various heterogeneous service types. In this way, GeSMO constitutes the underlying basis for the development of languages and tools which enable a developer to discover, invoke and compose heterogeneous services in a unified way.

As expected, the fundamental element in GeSMO is the notion of service. From a semantic and quality point of view **Fig. 2** illustrates the parts of a service that can be semantically annotated, as well as which of them can be quantified and thus have specific QoS properties.

Although GeSMO currently provides for the technologies of web, p2p, and grid services, it can be extended so as to support other types of services in a seamless manner. More information on the GeSMO model is provided in [5].

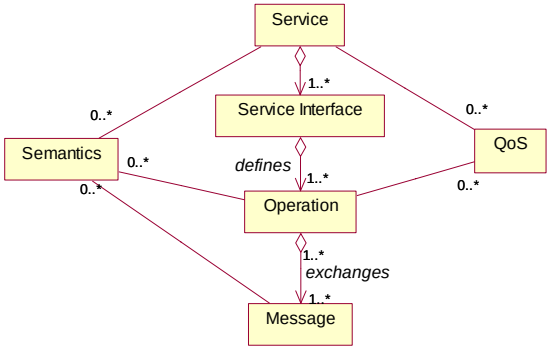


Fig. 2. Semantic and QoS point of view for a service

2.2 Visual Editor – Service Composition

The Visual Editor is the tool for modeling service compositions as VSCL graphs. The VSCL language is based on the Unified Modeling Language (UML) [3] and incorporates the necessary extensions to handle web services, p2p services, grid services, semantics, QoS chacteristics. The semantic modeling of services is an extension to UML provided by the SODIUM project. For the QoS notation VSCL is based on the use of the OMG's UML profile for modeling QoS and Fault Tolerance [2].

The first step in constructing VSCL graphs is to break down the composition into tasks, which can interoperate in order to finally achieve the overall goal. This composition of tasks is called an *abstract model* since there are no selected concrete services identified in this phase. This abstract model is used to search for appropriate candidate services to realize each of the abstract tasks. When services are selected for each abstract task, the result is a *concrete model*.

The concrete model can then be automatically translated into the lexical USCL language which can be executed by the SODIUM execution engine [4]. The next section explains how the USQL engine is used to discover appropriate services to register in the concrete model.

2.3 USQL and its associated Engine

The USQL Engine is used for the discovery of heterogeneous services (i.e. web, p2p, and grid services) over heterogeneous registries and networks, in a unified manner. As its name implies, the engine utilizes USQL (Unified Service Query Language), which is the SODIUM language for service discovery. USQL is XML-based that provides appropriate structures and elements to facilitate requesters in fully expressing their service requirements in a rich manner and, moreover, to cater for the formulation of the resulting service discovery responses. To accomplish this, the

language goes beyond the traditional syntax-based service requests and allows for the incorporation of semantics and QoS search criteria as well. It may occur, that service requests with so much information could potentially become rather vague, resulting in scrappy responses; to overcome such undesirable situations, USQL is equipped with a set of operators, which are categorized according to the type of requirement they apply to (i.e. syntactic, semantic, or QoS). These operators provide the means for effectively handling complex requests and capturing real-world requirements with accuracy and consistency.

The USQL Engine is a crucial component within the context of a service-oriented framework, facilitating the discovery of heterogeneous services that are used for solving an application-specific problem. Service discovery results are used to transform abstract workflow graphs conveying orchestrated tasks and their respective requirements into concrete service compositions that are then executed by the SODIUM execution engine. Alternatively, the USQL Engine may be used at run-time for the discovery of appropriate services to fulfill specific tasks within the service composition.

2.4 SODIUM Execution Engine

The SODIUM execution engine receives USCL documents containing the definitions of the service compositions and exposes an API for initiating, monitoring, and managing their execution. Service compositions can be exposed as web services [1]; hence, it is possible to access them from client applications through standard interfaces. For the invocation of constituent services the execution engine takes advantage of a set of plug-ins which cater for the invocation of web, grid and p2p services.

In the following we summarize research future challenges based on a general analysis and our experiences with the SODIUM platform.

3 Future research challenges

The following sub-sections present the research challenges that have been identified with respect to each of the SODIUM platform tools presented above. These challenges illustrate possible extensions that could be addressed in future version of the SODIUM platform.

3.2 Generic Service Model

Emerging trends in Software and Service Engineering as have been perceived by roadmap projects like the MAS Research Roadmap project [7] or NESSI (Networked European Software & Services Initiative) are pointing to a semantically enabled, ubiquitous and pervasive computing paradigm. Within this context, the Generic Service Model needs to be extended so as to anticipate the integration of additional types of services and concepts which support this emerging computing paradigm. Sensor and/or Actuator services are among the prominent types of services that need to be tackled, so as to leverage the interaction of service-oriented information systems

with their physical environment. In addition, the incorporation of agent services that are provided by software agents (mobile or not) or agent platforms will further leverage the convergence of the agent and the service oriented computing paradigms.

3.2 Service Composition

Although the Visual editor and VSCL language provide for the composition of heterogeneous services there are additional research challenges that need to be investigated. Some of these are the lack of support for interoperation (data transformations) and automatic composition.

The interoperation problem occurs when the developer in a composition wants to map the output from one service operation to the input of another service operation. Today, the developer must write such data transformations by hand. The lack of semantic annotation on existing services and their parameters make this work difficult. If, however, a set of mediation operations or agents were available, together with semantic annotated services, an inference engine could propose such data transformations.

Service composition requires carefully selection and structuring of service invocations in a particular order that solves a problem at a higher abstraction level. Eventually we would like to have the system itself proposing a composition for a particular problem. To facilitate automatic composition we need to increase the semantic descriptions of existing services and develop goal seeking inference engines.

Other areas which will benefit from more research are methods and techniques for modifying, verifying and monitoring execution of compositions taking into account quality of service parameters, run-time information, and service evolution.

3.3 USQL and its associated Engine

Within the context of the SODIUM project, both USQL and the USQL Engine have managed, more or less, to address some of the topics identified by NESSI. Yet, we plan to extend our current work to provide full support to most of the defined objectives.

As it has been also identified for the generic service model the USQL language as well as the USQL Engine will have to be extended so as to support the emerging device and sensor services. Sensor services impose a set of unique requirements due to their volatile nature and need to be further investigated so as to be supported by our approach.

USQL and the USQL Engine, besides integrating heterogeneous types of services enable requesters to enhance their queries with semantics. Given its abstract nature, the language supported semantic search criteria can be matched against any of the emerging standards (e.g. WSMO, OWL-S, WSDL-S). Thus, semantic interoperability is partially achieved in the course of service discovery. Yet, the lack of standard solutions for the semantic description of domains, and the existence of many overlapping ontologies and vocabularies for the same application domain render full semantic interoperability not feasible.

Other important issues which have an impact on the USQL engine and the USQL language are trust, security as well as QoS. Organizations and businesses need to be confident that the services they use are trustworthy as well as their quality properties

are within an agreed range. Thus, selected services should meet both functionality and security requirements, along with other QoS criteria, such as reliability. Although USQL provides a set of predefined QoS criteria and placeholders (i.e. extension points) for incorporating security requirements these issues in general remain open for future research in service discovery.

3.4 Execution Language

USCL along with its enacting USQL engine provide for the integration of web, grid and p2p services. Nonetheless, the range of services that are currently supported by the USCL language needs to be extended so as to address other types of services such as sensor or agent-based services.

Apart from supporting the integration of additional types of services, a research challenge where considerable research effort has been invested is the support for the development of dynamic and adaptable systems. Semantics as well as other existing approaches vie to facilitate the development of such systems, but up to now there haven't been any major advances. Another emerging approach that might be applied for the provision of such systems as well is DDDAS (Dynamic Data Driven Application Systems) [6] which is currently gaining momentum.

Composition languages have also received considerable criticism regarding the incorporation of Quality of Service properties (QOS) and requirements during the design as well as the execution of composite services [8]. Emerging systems need to be able to retain a level of QoS by selecting and composing constituent services with specific quality properties and by claiming appropriate systems resources according to the end users' needs.

Last but not least, some other important issues that need to be addressed by the existing orchestration languages include the handling of data and persistence [8]. The composition of heterogeneous services such as grid services or sensor services needs to be supported by appropriate constructs that will be able to facilitate the exchange of high volumes of data that are produced or consumed by such services. Hence, composition languages as well as their supporting execution engines need to be extended so as to accommodate advanced data handling constructs.

4 Concluding Summary

Service oriented development (SOD) is a new trend in software engineering. SOD is already affecting the development of business oriented systems turning them into service compositions. However, the heterogeneity in protocols and standards of existing service types is a major obstacle for the discovery of services and their integration in service compositions.

In this paper we briefly described a platform called SODIUM which provides tools, languages and related middleware for supporting the whole lifecycle of service-oriented applications (i.e. from requirements modeling to their execution) composed of heterogeneous services. Specifically, SODIUM supports abstract as well as concrete modeling of service compositions (by providing the VSCL language and

editor), unified discovery of constituent heterogeneous services (through USQL and the query engine) and execution of service compositions (through the USCL Engine).

The open and extendable architecture of SODIUM doesn't alter the underlying protocols and infrastructure used by the various services, but rather hides the specific details from service composition developers. Furthermore, besides the service types currently supported, i.e. web, grid and p2p, SODIUM provides for the easy integration of any other service type. We consider the SODIUM platform to be a suitable basis for addressing a number of future research challenges described in section 3.

Acknowledgement. This work has been partially supported by the European Commission under the contract IST-FP6-004559 (project SODIUM: Service Oriented Development in a Unified fraMework).

8 References

1. T. Heinis, et.al, ***Publishing Persistent Grid Computations as WS Resources***, In: Procs. of the 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, December 2005.
2. OMG, *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*, OMG Final Adopted Specification, ptc/04-09-01
3. OMG, "*Unified Modeling Language: Superstructure, version 2.0*", OMG Final Adopted Specification, ptc/04-10-02
4. David Skogan, et.al "*D9-Detailed Specification of the SODIUM Service Composition Suite"*, SODIUM (IST-FP6-004559) project's deliverable, June 2005
5. A. Tsalgatidou, et.al. "D4-Generic Service Model Specification", SODIUM (IST-FP6-004559) project's deliverable, June 2005
6. Frederica Darema, "Data Driven Applications Systems: New Capabilities for Applicaiton Simulations and Measurements", In Procs. of ICCS 2005, Atlanta, Georgia, USA, May 22-25, 2005
7. Michael N. Huhns, et.al. "Research Directions for Multi Service-Oriented Multiagent Systems", IEEE Internet Computing, Nov.-Dec. 2005, pp 65-70
8. Aleksander Slominski "On Using BPEL Extensibility to Implement OGSI and WSRF Grid Workflows", Presented in Workflow in Grid Systems workshop in GGF-10, March 9th 2004, Berlin, Germany

Research Challenges in Mobile and Context-Aware Service Development

Julien Pauty, Davy Preuveneers, Peter Rigole, Yolande Berbers *
*Department of Computer Science, K. U. Leuven
B-3001 Leuven
Belgium

Abstract. *In this paper we present several major research challenges in mobile and context-aware service development. We present how we tackle these challenges through two of our research projects: CoDAMoS (Context-Driven Adaptation of Mobile Services) and CROSLOCIS (Creation of Smart Local City Services). We contribute to the following five challenges: (1) context-aware adaptation of mobile services; (2) service relocation; (3) service description and discovery; (4) security in service architectures; (5) management of context data.*

Keywords: Context-aware services, context-aware middleware

1 Introduction

With the need for more flexibility and adaptability the IT industry is shifting from making products to providing services. A service typically involves two participants: the service provider and the service consumer. A service is provided upon the request of the consumer.

The advantages of the service approach include:

- Services can be requested by an external entity, provided that the service interface is public. For example, Amazon [Ama] and Google [Goo] already provide services to external entities.
- Service composition: several services can be composed by an entity to provide an enhanced service to other entities.

A software architecture which relies on the service approach is commonly called a service oriented architecture (SOA). SOAs comprise loosely coupled, highly interoperable application services . These services interoperate based on a formal definition independent of the underlying platform and programming language. A SOA is independent of development technology (such as Java and .NET). The software components become very reusable because the interface is defined in a standards-compliant manner. In this way, for example, a C++ service could be used by a Java application. [1].

[1] Definition of SOA taken from Wikipedia : http://en.wikipedia.org/

The service industry is currently facing several challenges [SM], such as:

- How to develop new services? This challenge includes the traditional problems of software engineering. More specific to the service industry, it also includes :
  1. Service description: how to describe the service from the service developer point of view and from the service consumer point of view?
  2. Service discovery: how to discover new services, matching the users needs.
  3. Service monitoring: how to monitor service behavior, in order to ensure that it matches service specification?
  4. Service composition: how to compose a new service on the basis of several existing services in order to create a new service?
- How to provide services that leave the user in control? This challenge addresses the compromise between automatic and user controlled service provision.
- How can different kinds of services inter-operate with each other and on different kinds of networks?

In this paper, we focus on mobile and context-aware services. In their simplest form, mobile services are traditional services delivered via mobile devices, such as mobile phones or PDA's.

Mobile services can also be specifically tailored to the needs of mobile users. A context-aware mobile service is adapted to the current situation of the user. The goal of a context-aware service is to support the user by providing him with the right service at the right moment. If the user context changes, the context-aware service should self-adapt or be adapted to the new context. A context-aware service is autonomous and tries to support the user without too much interactions with a computing device.

The creation of mobile and context-aware services raises new research challenges, including:

- Service adaptation: mobile and context-aware services must be provided on several kinds of devices, ranging from wall displays to mobile phones. These devices have different resources, such as screen size or memory, so services must be adapted to the available resources on the running host.
- Context-awareness: the services must be aware of the current context of the user and self-adapt to user context changes.

In this paper, we present several major research challenges of mobile and context-aware service development. We present how we address part of these research challenges in our current research projects. Our research projects are mostly dedicated to the creation of methodologies, middlewares and infrastructures to develop and execute mobile and context-aware services.

Section 2 presents the CoDAMoS projects and the associated research challenges. We continue in section 3 with the presentation of the CROSLOCIS project. Before concluding, we present in section 4 two related research projects.

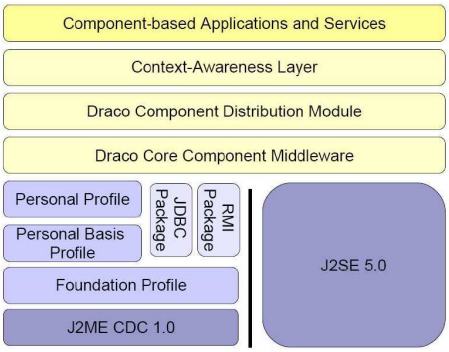


Figure 1: Overall structure of our context-aware middleware

2 CoDAMoS: context driven adaptation of mobile services

In CoDAMoS [Cod], we work on innovative and generic software methodologies and techniques to support the context-driven adaptation of mobile services. The developed system enables any service to detect changes in the user context and to dynamically adapt the services to this new context.

In this section we first describe our context-aware middleware. Following sections describe the research challenges addressed by this middleware : (1) service description and service discovery; (2) service adaptation; (3) service relocation; (4) context aware adaptation.

2.1 Context-aware middleware

Our context-aware middleware [PVRB05] includes: a context-awareness layer, a distribution module and Draco [VRUB03], which is our component oriented middleware (see 1). Draco is based on Java and available on devices running J2ME and J2SE. Draco enables the service developer to create services by connecting several component together. The distribution module enables transparent relocation of the whole service or a component of the service on a remote device. The context-awareness layer enables the service to detect changes in the context of the user and to initiate service relocation on nearby devices. The context-awareness layer can also dynamically replace components with lighter versions to save resources.

2.2 Service description and service discovery

Service description and service discovery are two linked challenges: the description of a service is used during service discovery. These research challenges are not specific to context-aware mobile services.

In the context of ambient intelligence service discovery enables us to discover services that match the user needs. A service can be found in the vicinity of the user on the nearby devices or on a distant host. In CoDAMoS we focus on service located on the nearby devices.

In CoDAMoS, services are described using an ontology which is an extension of the OWL-s ontology

[PdBW+04]. This ontology enables the service developer to define:

- the required resources by the services;
- contracts, which provide guarantees on the service behavior;
- runtime adaptation, by defining for each component alternative components or whether this component is optional.

Service discovery is done by matching a service request with the service description.

2.3 Service adaptation

In CoDAMoS, we are targeting ambient intelligence services. Ambient intelligence implies that the user is surrounded with different kinds of devices, ranging from mobile phones to desktop computers. Ambient intelligence implies that services must be able to run on these different target devices. However, these devices have different resources limitations and input-output capabilities, so services must be adapted to these device-specific constraints.

To reduce the costs of service development, we rely on a component-based methodology. With this methodology, a service is composed of several connected components. A component is a software black box that performs a specific function. Components are composed by means of their interfaces: they can provide interfaces to and require interfaces from other components. Component interfaces are reified into ports. Two components can be connected through ports, if the corresponding interfaces match. We can consider that a component can provide and request services through its ports.

Component-oriented design is usually seen as a solution to code reuse. In the context of ambient intelligence, component-oriented design offers a simple and elegant way to develop services that can be quickly adapted to several kinds of devices. With our methodology, the service developer specifies for each components if it is optional or mandatory for the service. The developer can also specify alternative versions of the same component, each version using a different amount of resources. For example, on a resource constrained device, optional components may not be deployed, or a down-sized version of a component may be used to save resources. Thus, it is possible to adapt the service to different devices, while keeping the same global software structure for the service.

To support the development and deployment of component oriented services we have developed Draco, which runs both on mobile devices and desktop computers.

2.4 Service relocation

In the preceding section we have seen that using a component oriented methodology it is possible to develop several versions of the same service, each version fitting with the available resources of different devices.

In the context of ambient intelligence the user is always surrounded with different computing resources. To give the best quality of service, it may be needed to relocate the service to another device. For example, when the user leaves his office, its current application is relocated on his mobile device. In this case, the mobile device has different capabilities compared to his desktop computer. Therefore, while

relocating the application on the mobile device, it is necessary to adapt it to the capabilities of this device. In this case we perform a runtime adaptation.

To support runtime adaptation of services, Draco enables runtime component update and service relocation. In this way, it is possible to disconnect optional components or replace one or more components with alternative components, while the service is relocated on another host.

Draco also enables the relocation of only one or more components on another host. For example, if the user is close to a free wall display, the display component of the service he is currently using can be relocated on the wall display to improve his experience. Component relocation is performed by the distribution module [RVL+05].

2.5 Context-aware adaptation of services

Once services are developed and ready to be adapted to different target devices we need a mechanism to detect when service migration is needed and how it must be done. This is the role of our context-awareness layer.

We have developed a context-awareness layer to enable a service to self-adapt to changes in the users context. The users context is modeled using an ontology [D. 05, PVRB05]. Relying on this ontology, applications can define adaptation rules. For example, consider the rule: if the battery level is below 20% and if the user is close to a desktop computer then the service must be moved to this computer. Using the facts corresponding to the user location and the battery level, this rule enables the service to determine if it must move to another host or not.

Using this ontology the service developer can create rules which specify when the service must be adapted. It is also possible to instantiate rules at runtime. For example, the user might specify in his preferences that he never wants to move the service, or that the service must always try to move to a nearby desktop computer.

The context-awareness layer is itself built up of components. Thus, services can get context-awareness capabilities by simply connecting their components to the context components.

3 Crolocis

CROSLOCIS [Cro] is a new basic research project dedicated to the creation of a service architecture where new innovative local mobile services can easily be created, deployed and consumed by mobile users. The overall goal of this architecture is to lower the threshold for SME to set up their own mobile services and allow charging from the content or service provided in business-to-consumer or business-to-business scenarios.

The CROSLOCIS project is complementary to CoDAMoS, in the sense that its focus is broader, i.e. mobile services, and not just ambient intelligence environments. The focus of CROSLOCIS also differs from the focus of CoDAMoS: where CoDAMoS is looking at methodologies and techniques for adaptation, CROSLOCIS is focusing on architecture.

In the CROSLOCIS project, the Distrinet group will address two main research challenges:

1. Security: in a service platform, security is needed to authenticate the user when he requests a

service, and to ensure the non-repudiation of the charging contract between the user and the service provider.

2. Collection and distribution of context data: context data will enable service providers to tailor their services to the customer needs and preferences. However, this data is privacy sensitive, so the service platform must have mechanisms that enable the service providers to access context data while preserving the privacy of users.

In this section we present each of the two aforementioned challenges.

3.1 Security architecture

The main objective of the security architecture is to provide an access control and non-repudiation solution for the service architecture. This security architecture should rely on the Belgian electronic identity card (e-id). Currently, the e-id has a classical bankcard format and contains two public private key pairs and the corresponding certificates, one for authentication signature, and one for non repudiation. These certificates and the PKI back-end infrastructure are not only used by the government, but can also be used by commercial parties.

Work on the security architecture will start by studying the limitations of the existing PKI infrastructure used in e-id. One of the possible evolution of the e-id could be to extend its scope and use it in private PKI infrastructures such as banking applications, ATM and internet e-commerce.

Extending the scope of the e-id raise several challenges. One of the challenges comes from the physical form of the e-id. Indeed, in the envisioned city services, the e-ID card will be physically needed for multiple devices at the same time. For example, if you need the card to logon to your PC, you need to take it out to make a phone call with a dedicated mobile device. Therefore, we will have to investigate possible ways to split the physical authentication function of the card on various devices, and the impact of this splitting on the overall security.

3.2 Collection and distribution of context data

In CoDAMoS context was used as a tool to trigger adaptation of services. In CROSLOCIS, context will be used to provide enhanced services, typically services based on the user location and preferences.

Collection of context consists in capturing, storing and reasoning on the user context. In CROSLOCIS, we will use all the available sources of context information. To capture the user context, we will particularly rely on the information provided by the user terminal, but also on information provided by the user profile and the network, such as the cell-id. The user context will be modeled with an ontology. This ontology must encompass all the envisioned city services and remain open for future services. Using a reasoning system and this ontology we will be able to infer user context.

In CROSLOCIS, context distribution means that the service platform will enable third-party service providers to create enhanced services by allowing them to access context data. However, context data is privacy sensitive and several service providers may access to this data through the service platform. Therefore, to address context distribution, we will have to develop mechanisms that enable service

providers to access context data without breaking users privacy. These mechanisms will have to prevent several service providers to enlarge their view on the users profile by sharing and aggregating their respective knowledge on the users. The mechanisms will also have to prevent unwanted propagation of context-data among service providers, typically when several services are chained among different service providers.

4 Related projects

In this section, we briefly present two related research projects1 that are also investigating the research challenges of mobile services.

The Amigo [Ami] project is dedicated to ambient intelligence for networked home environment. Home networks start to emerge, but the lack of interoperability between devices, the complexity of configuration and the absence of compelling services prevent it to really change people's live. To improve the usability and attractiveness of home networking for the end-user, Amigos main objective is to research and develop open, standardized, interoperable middleware and intelligent user services for the networked home environment, which offer users intuitive, personalized and unobtrusive interaction by providing seamless interoperability of services and applications. Amigo clearly addresses the service creation challenge, with a strong focus on context-aware services. The Amigo middleware also enforces security and privacy policies.

The MADAM [Mad] project is strongly related to CoDAMoS. The overall objective of MADAM is to provide software engineers with modeling language extensions, tools and middleware to create mobile and adaptive applications. The MADAM middleware relies on two main technical elements: (1) components to monitor context and take decision about adaptation; (2) dynamic reconfiguration based on component architecture and reflection. Based on these technologies the middleware is able to select the service that best fit the user requirements and current context.

5 Conclusion

In this paper we have presented several research challenges of mobile and context-aware service development, through the research projects we are participating to. We are contributing to the following research challenges:

- Context-aware adaptation of services: the CoDAMoS middleware enables services to self-adapt when the user context change.
- Service relocation: the CoDAMoS enables a service to migrate on a nearby host when the available resources drop, to provide the best user experience.
- Service description and discovery: the CoDAMoS middleware relies on an extension of the OWL-s language to describe services. Services are discovered by matching a service request with its description.

- Security architecture: in CROSLOCIS we will develop a security architecture for user authentication and non repudiation of charging contracts between service providers and users.
- Management of context data: in CROSLOCIS we will develop mechanisms to enable service providers to access context data while preserving the users privacy.

References

[Ama] The Amazon Web Services. http//www.amazon.com/gp/aws/landing.html.

[Ami] The Amigo project: Ambient intelligence for the networked home environment. http://www.hitech-projects.com/euprojects/amigo/.

[Cod] The CoDAMoS project. http://www.cs.kuleuven.be/ distrinet/projects/CoDAMoS/.

[Cro] The Crolocis project. http://www.ibbt.be/site/index.php?id=160&L=1.

[D. 05] D. Preuveneers and Y. Berbers. Adaptive context management using a component-based approach. In *International Conference on Distributed Applications and Interoperable Systems (DAIS 2005)*, June 2005.

[Goo] The Google API. http://www.google.com/apis/.

[Mad] The MADAM project. http://www.ist-madam.org/.

[PdBW+04] D. Preuveneers, J. V. den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In *European Symposium on Ambient Intelligence (EUSAI 2004)*, 2004.

[PVRB05] D. Preuveneers, Y. Vandewoude, P. Rigole, and Y. Berbers. Middleware Support for Component-Based Ubiquitous and Mobile Computing Applications. In *International Middleware Conference (Middleware 2005) (demo)*, 2005.

[RVL+05] P. Rigole, C. Vandervelpen, K. Luyten, Y. Vandewoudeand K. Coninx, and Y. Berbers. A Component-based infrastructure for Pervasive User Interaction. In *Workshop on Software Techniques for Embedded and Pervasive Systems (Pervasive 2005 workshop)*, 2005.

[SM] A-M. Sassen and C. Macmillan. The Service Engineering Area: an Overview of it Current State and a Vision of its Future. http://www.cordislu/ist/st/serv_eng.htm.

[VRUB03] Y. Vandewoude, P. Rigole, D. Urting, and Y. Berbers. Draco : An adaptive Runtime Environment for Components. Technical Report CW 372, Dept. of Computer Science, K.U. Leuven, 2003.

Context Management and Semantic Modelling for Ambient Intelligence

Fano Ramparany*, Jérôme Euzenat,**
Tom Broens*, **Jérôme Pierson***,
André Bottaro*, Remco Poortinga ***
* France Telecom R&D
** INRIA Rhône Alpes
*** Centre for Telematics and Information Technology
**** Telematica Instituut

Abstract. *Ambient Intelligence aims at pushing forward a user centric vision of Pervasive Computing, where the environment better serves our need. This paper describes our current work on modelling and managing context information for smart environments.*
Keywords: Context-Awareness, Management, Modelling, OWL

1 Introduction

Ambient Intelligence (AmI) builds upon concepts from the Pervasive Computing (PC) paradigm. PC aims at flooding our daily physical environment in computing and communication in such a way that the environment can act as an interactive collection of interconnected network of 'daily things'. AmI aims at pushing forward a vision where this environment also proactively serves our needs by understanding our activities, anticipating our needs and collaborating with us in achieving our daily tasks. To make this vision come true, the AmI environment needs to be aware of any information that is helpful for identifying user's activities, needs and tasks. This information is to be found in the physical environment as well as from the users themselves or from the computer systems they use. We call such kind of information 'contextual information'.

In this paper, we introduce an infrastructure for managing context information that we are developing in the Amigo[1] project. We state the main problems we are addressing while designing this infrastructure. We illustrate its role through concrete scenarios. We then introduce our architecture for the Context Management infrastructure. We then present the current state and preliminary results of our on-going work. We finally discuss how a service-oriented approach could exploit the context management infrastructure to give rise to context aware services and conclude.

[1] The authors would like to thank the CEC for partially supporting the work reported here, in the framework of the IST-Amigo collaborative project.

2 Problem Statement

The term 'context' is overloaded with a wide variety of meanings, depending on application purposes and on the research community standpoint ([1]). Several research communities like Information bases, Artificial Intelligence, Human Computer Interaction and Ubiquitous Computing have proposed ‘context’ definitions. We adopt the general definition proposed by Dey ([2]): "…Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves…"

In ambient intelligence, we can distinguish three categories of context ([3]):

- Device context like memory, computation power, networks (and their quality), codecs, etc.
- User context divided into personal (health, mood, activity, etc.), applicative (emails, visited websites, preferences, etc.) and social (relatives, employment, activism, etc.) contexts.
- Physical context: contextual information related to the physical environment of an entity (device, room, building, and user). Examples are location, time, weather, altitude, light.

To be able to use contextual information in an AmI environment, we need some management functionalities. Context management is responsible for propagating context information from sensors to the application, storing it, controlling various manipulations on it (e.g., aggregation), and providing access control to the context information.

2.1 CONTEXT AWARENESS

Context awareness denotes the use of contextual information in computer system functionality. According to Dey ([2]), "Context-awareness is the property for a system of using context to provide relevant information and/or services to the user, where relevancy depends on the user’s task".

2.2 CONTEXT AWARENESS SCENARIOS

In this section, we introduce 3 simple scenarios, which illustrate the concept of context awareness applied to ubiquitous computing services. Context awareness here means that the primary function of the service adapts to the current context of the physical object, i.e. the context that holds while invoking the service.

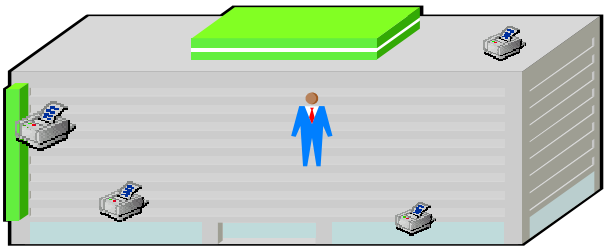


Figure 1: Printer location scenario.

In the printer location scenario, the user (precisely the printing service) benefits from the printers' physical location information and his/her own location information. The service offers printing facilities minimizing the moves throughout the building.

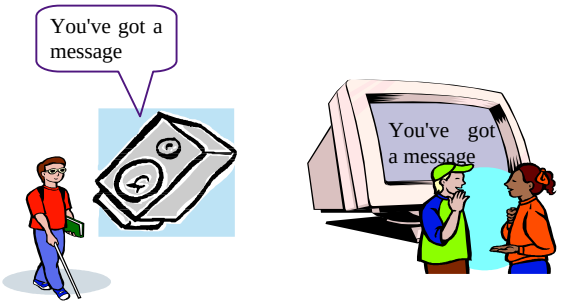


Figure 2: Profile aware notification scenario.

In the recipient's profile aware notification scenario, a high level service aims at notifying a user that "he/she has just received a mail". The service takes the personal context of the user (e.g. accessibility profile – deaf/blind/) into account to select the most appropriate notification modality (voice/sound notification service/device or text/screen notification service/device).

Figure 3: Energy-preserving bulb scenario

In the energy preserving bulb simple scenario, the context aware bulb adjusts the light intensity to the environmental context (e.g. darkness/sunshine) or user context (e.g. activity).

3 Approach

Scenarios described above quite naturally suggest the use of a central context management service, designed to supply end user services, applications, or devices with the context information they need. The main advantage of this approach is to gather contextual information at a single place, making it easier to find and retrieve context information. The main drawback of a central context management service is the centralized approach; which could even be inconsistent with the very notion of context, as it sets context as task insensitive (as perceived by client services). On the opposite, a purely decentralized management fails in federating resources and tends to overload sensor devices.

Our solution is to define services in such a way that they become capable of managing their own context, and let some of them supply other devices and services with context information. This last configuration makes it possible to aggregate context information. Instead of implementing context management as a mere central service, we prefer to distribute it over devices and application services, as software components. In the following sections,

we introduce the architecture of the context management infrastructure and focus on one of its main components: Context information modeling.

3.1 CONTEXT MANAGEMENT INFRASTRUCTURE

Context information is managed in a distributed way so that it is shared among interested entities. Devices (e.g. sensor devices) provide low-level context information. Context-aware services consume information coming from devices and other services in order to build their own high level context information which they may in turn offer to other services. None of these entities holds a complete global view of context information. Figure 4 displays how services and devices exchange context information.

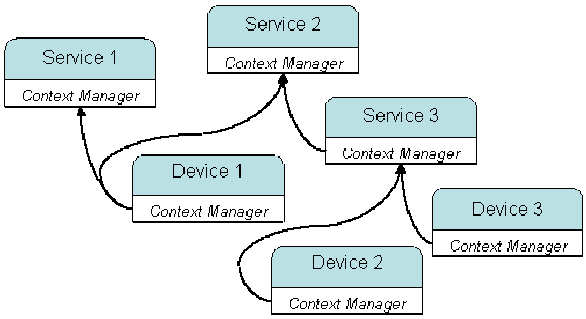


Figure 4: Context information flow.

Every entity (i.e. device or service) contains parts of the functionality of the whole context management infrastructure, denoted as 'Context Manager' blocks in figure 4. The different functionalities that can be part of these blocks are:

- Context Sources providing context information to any interested party. Two types of Context Sources can be distinguished:
  - Context Wrappers adapting raw data coming from sensors into the context model.
  - Context Reasoners interpreting context queries and aligning heterogeneous models.
- Context Brokers keeping track of the different context sources and their information.
- Context Stores storing information used to satisfy incoming queries. Each entity holds its own context model according to a specific ontology.

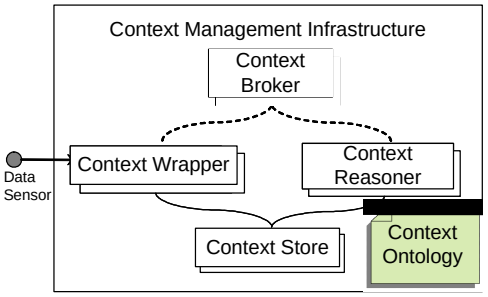


Figure 5: Context Management Infrastructure functional elements.

There are three complementary ways to restore interoperability between ontologies:

- Standardizing the required ontologies a priori.
- Recording correspondences as soon as ontologies are made available by parties.
- Dynamically matching ontologies.

In every case, the infrastructure must be able to provide the best possible use of available context information. To this purpose, we use and adapt technologies developed in the framework of the Semantic Web [4].

3.2 CONTEXT INFORMATION MODELING

Developing context-aware services requires an appropriate way of representing context information, building it incrementally, maintaining/updating it over time, and being able to retrieve it dynamically. We address the representation issue in this section.

Based on the analysis of previous work ([5]), we base our model of context information on anontology of physical context. The latter defines concepts, i.e. the types of the relevant objects in the studied context. For instance, temperature, weight, and distance are concepts of the context ontology. Every concept is defined in terms of attributes, attribute constraints or restrictions, and relations with other concepts. For example the temperature concept has a numerical attribute "value" which is restricted to be higher than -273.15ºC and it shares the relation "characterizes" with the concept "place". The context ontology could be viewed as an abstract model. Context information is then modeled as instances of this ontology.

Primary types are modeled in the first version of the infrastructure. They include low level concepts such as discrete, scalar, continuous, array data (Figure 6). Higher level concepts and relations which form the context ontology will be integrated in subsequent versions. Such incremental approach via composing, completing and consolidating preliminary models suit well the process of building ontologies.

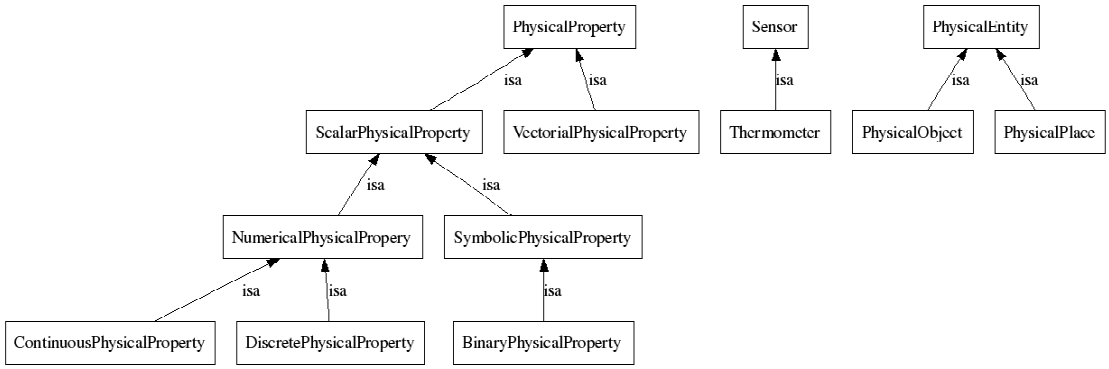


Figure 6: Physical Context model excerpt.

Sensor and PhysicalEntity class hierarchies attach context information to physical places or objects, and relate context information to its source (Figure 7). Context information is made

of instances of PhysicalProperty's terminal subclasses. For example, if a Thermometer named Thermometer_1 provides a temperature measurement of 25.0ºC, an instance of ContinuousPhysicalProperty class called Temp_measurement_1 is created, the number 25.0 is assigned to the slot "Value" and a relation "measured_by" is established between this instance and the sensor (Figure 7). Generic data properties like precision, reliability, timestamp and resolution are defined at the topmost level of the Physical properties hierarchy.

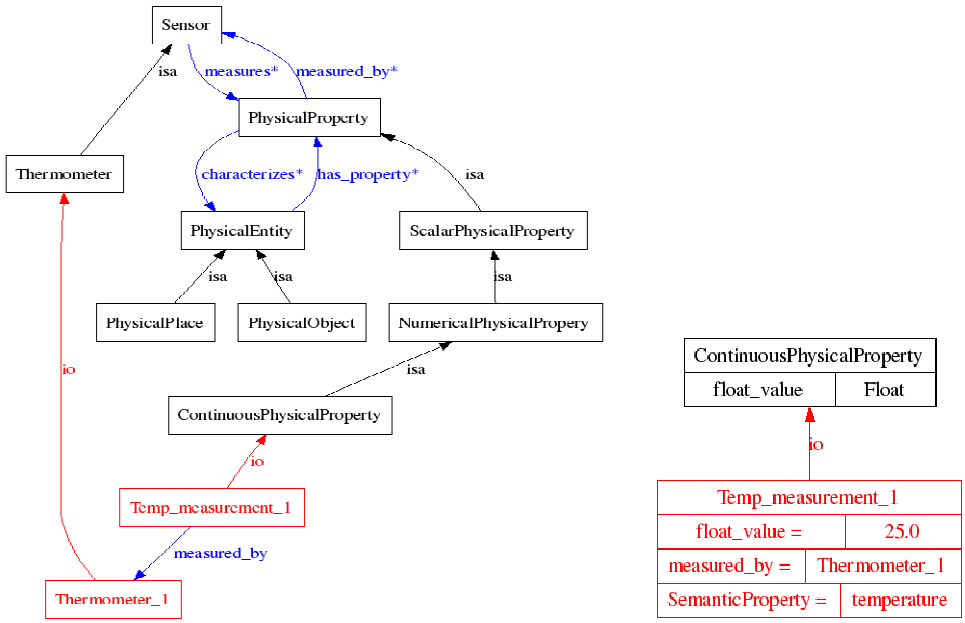


Figure 7: Instance of context representation.

3.3 CONTEXT QUERYING AND REASONING

Context information inference makes it possible to derive implicit context information from explicit one which is present in the context store. For example, in order to know whether a room is lightened although there is not any sensor assessing this environmental property, we might derive this information from the time of the day and the status of the shutters (open/closed). The reasoner provides support to context information inference. Our reasoner is built upon the Jena2 system ([6]). Jena2 includes a generic rule based inference engine together with configured rule sets for RDFS and for OWL languages.

4 Discussion

The presented work is still in progress. So far, we found it very convenient to rely on foundation technologies of the Semantic Web and Web Services. The main advantages are:

- Flexibility: we have redesigned the abstract model of context information (context ontologies) many times. These changes induced insignificant modifications on the

implementation of application services and sensor wrappers. Only the formulation of context information queries is impacted.

- Sharing and Reusability: some ontologies of context information are available "off the shelf" (e.g. location context). Current project is developing specific ontologies on time and geographical information. Such ontologies could be easily reused or adapted to fit our need.
- Openness: adopting Semantic Web will foster the accessibility of "local", "physical" services in pervasive computing from "virtual", "disembodied" Web services, and vice-versa.

In the very short term, we will come up with an operational version of a context-aware service infrastructure which will exhibit the following features:

- Context aware service lookup. Context information is used to complement service requests at the client side (e.g. use the location of a user to find the nearest restaurant), or at the server side (e.g. use the current date to filter out closed restaurants).
- Proactive services: Context information will be used to trigger specific services as predefined context information configurations occur.
- Context dependent services: This is the "conventional" use of context, where services adapt their behaviour by posting context queries and adjusting their computation based on the replies they receive to their queries.

With respect to existing work, as other research teams ([7], [8]) we aim at developing a generic context management environment suitable to the ubiquitous communications world. However we have departed from a centralized solution which we believe fails in capturing dependency of context on the task to achieve or on the service it will alter.

Adopting a decentralized solution where context description is to be shared among its consumers (services) and its producers (services or sensors) requires a semantic modeling of context information. Such semantic modeling is necessary to cope with the heterogeneity of the resulting partial models. Some work such as the CoBra system ([9]) has already addressed this issue, however with the limitations incurred by a centralized approach.

We interestingly pursued a path similar to Crowley et al. ([10])’s with the concept of "contextors". We adopted the Semantic Web ontology language OWL, to model context information. We believe this will facilitate interoperability with the Web Services world.

5 Conclusions

In this article we have described our approach for modeling and managing context information in a service oriented framework for Ambient Intelligence environments. This work is conducted within the IST-Amigo project ([11]). So far, our efforts have focused on context modeling and reasoning and our results reveal promising.

We still have to tackle the integration of our work into a complete service infrastructure which is developed in parallel within the Amigo project, and the assessment of the infrastructure in several real world applications.

We also plan to conduct additional work on context management, dealing with the:

- Merging and aggregating contextual information
- Learning of context and context descriptions
- Interoperability mechanisms with external context information sources, such as legacy mobile location servers

References

[1] Theodorakis, M., Analyti, A., Constantopoulos, P. and Spyratos, N. (1998) Context in information bases, in Intelligent agents, in the proceedings of the third IFCIS Conference on Cooperative Information Systems (CoopIS'98), New Yorks.

[2] Dey, A., Salber, D. and Abowd, G. (2001) A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. in Human-Computer Interaction vol. 16. p. 97-166.

[3] Schilit, B, Adams, N. and Want, R. (1994), Context-Aware Computing Applications IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, USA, 1994.

[4] Euzenat, J. (2004) An API for ontology alignment. Proc. 3rd ISWC, Hiroshima (JP), LNCS 3298:698-712.

[5] Flury, T., Privat, G., and Ramparany, F. (2004) OWL-Based location ontology for context-aware services in Proceeding of the worshop on Artificial Intelligence in Mobile Systems. Bristol - UK. p. 52-58.

[6] www.hpl.hp.com/semweb/jena2.htm.

[7] Gray, P and Salber, D. (2001) Modelling and using sensed context information in the design of interactive applications, in Proceedings of the 8th IFIP working conference on engineering for human-computer interaction (EHCI'01), Toronto, Canada, May 2001.

[8] Dey, A. (2000), Providing Architectural Support for Building Context-Aware Applications PhD Thesis, College of Computing, Georgia Institute of Technology, Dec. 2000

[9] Chen, H., Finin, T. and Joshi, A., 2003, The Role of the Semantic Web in pervasive Context-Aware Systems, Proceedings of ISWC 2003

[10] Crowley, J.L., Coutaz, J., Rey, G., and Reignier, P. (2002) Perceptual components for context aware computing. in International Conference on Ubiquitous Computing. Göteborg, Sweden. p. 117-134

[11] http://www.hitech-projects.com/euprojects/amigo/.