CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

le cnam

ÉCOLE DOCTORALE INFORMATIQUE, TÉLÉCOMMUNICATION ET ÉLECTRONIQUE (EDITE - PARIS)

ÉQUIPES VERTIGO - LABORATOIRE CEDRIC

THÈSE DE DOCTORAT

présentée par : **Emilian PASCALAU**

soutenue le : **7 avril 2014**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline / Spécialité : **Informatique**

Vers un développement Web orienté utilisateur

Towards Web User-Centric Development

THÈSE DIRIGÉE PAR

M. RIGAUX Philippe	*PR, CNAM*

RAPPORTEURS

M. GROSS-AMBLARD David	*PR, Univ. Rennes 1*
Mme. GRIGORI Daniela	*PR, Univ. Paris-Dauphine*

EXAMINATEURS

Mme. BENBERNOU Salima	*PR, Paris Descartes*
M. TRAVERS Nicolas	*MdC, CNAM*
M. ZAMFIROIU Michel	*Directeur, KarmicSoft inc.*

To my parents Emil and Iuliana

Aknowledgements

I feel obliged to first thank God for the health and strength He gave me through out the years of study.

Second I would like to thank my supervisor Prof. Philippe Rigaux, for all his support both academic and moral, without whom I would have never finished my thesis.

Also I would like to thank Dr. Adrian Giurca for our discussions and arguments on the topic.

A special thank to my dad for his moral support and to all my friends whom I am not going to name here, but whom in one way or another helped me to get here.

A most sincere Thank you!

Contents

Résumé **iii**

Abstract **v**

Résumé étendu **vii**

1 Introduction **1**
1.1 Objective of this thesis . . . 2
1.2 Our approach . . . 3
1.3 The GPS device metaphor . . . 4
1.4 Contributions . . . 5

2 Related Work on Mashups **9**
2.1 Mashups and Web Services / SOA . . . 10
2.2 Semantic Web and Web of Data . . . 12
2.3 Mashups and Software as a Service . . . 12
2.4 Mashups and Portals . . . 14
2.5 Mashups - Conceptual approaches . . . 17
2.5.1 Mashups as a Collection of Widgets . . . 17
2.5.2 Pipes Based Mashups . . . 22
2.5.3 Hybrid . . . 24
2.5.4 Domain Specific Languages for Mashups . . . 29
2.6 Discussion . . . 30

3 A User-centric approach. Conceptual Model **35**
3.1 Our proposal - A User-centric approach . . . 36
3.1.1 End-user / user-centric . . . 36
3.1.2 Two layer system . . . 36
3.1.3 Plan . . . 37
3.1.4 Intelligent system and human user and system interacting with each other . . . 39
3.1.5 The approach . . . 40
3.2 Conceptual Model . . . 40
3.2.1 Concept . . . 41
3.2.2 Context . . . 43
3.2.3 Behavior . . . 44
3.2.4 Mashup . . . 46

4 Architecture **51**
4.1 Our Approach vs. GPS metaphor . . . 51
4.2 TomTom . . . 53
4.2.1 Digital maps - The plan . . . 53
4.2.2 How the GPS System Works . . . 54
4.2.3 Outcomes . . . 54
4.3 The Architecture . . . 54
4.4 Discussion . . . 59
4.4.1 Web pages as Web services . . . 59
4.4.2 Mashups styles . . . 60
4.4.3 DOA and SOA influences . . . 61

5 Execution **63**
5.1 DOM and DOM events . . . 63
5.2 The Rule Engine . . . 63
5.3 Rules . . . 66
5.3.1 Event . . . 68
5.3.2 Conditional elements . . . 69
5.3.3 Actions . . . 71

6 Implementation **73**
6.1 The prototype . . . 73
6.2 Use Cases . . . 76
6.2.1 Conference Calendar - The Recurrent Use Case . . . 76
6.2.2 Security - Web Policies . . . 80
6.2.3 Personalization . . . 80
6.2.4 Web Analytics . . . 82
6.3 Requirements . . . 82

7 Conclusions **85**
7.1 Future Work . . . 86
7.1.1 Visual Modeling . . . 86
7.1.2 Mobile Platforms . . . 87
7.1.3 New Reasoning Techniques . . . 87

Bibliography **87**

Résumé

World Wide Web (WWW) est devenu le plus grand dépôt d'informations que l'homme ait jamais assemblé et il est en croissance continue. WWW s'est transformé en un environnement génératif qui favorise l'innovation par le développement des technologies et par un changement dans la perception des gens sur le Web et comment l'utilisent. Le nouveau WWW ou l'Internet de l'Avenir est celui d'un Internet des Services et un Internet des Objets.

Naturellement, une série des questions se posent à partir de ce contexte : comment filtrez-vous les choses pour créer plus de valeur que vous obtenez actuellement ? Comment pouvez-vous regrouper les choses d'une manière intelligente et facile au lieu de la faire dans votre tête? Le monde ne peut pas être décrit sans ambiguïté, alors comment pouvez-vous permettre aux utilisateurs de traiter avec le monde à leur manière, en fonction de leur compréhension? Levine dans son livre "Cluetrain manifesto" a argumenté que les marchés sont conversations, alors comment peut-on impliquer les utilisateurs dans la conversation ? Comment les utilisateurs peuvent être autorisés à la consommation facile des services, de l'information, des choses qu'ils trouvent autour?

Cependant, la conception et le déploiement d'un tel logiciel capable d'interaction directe et l'autonomisation de l'utilisateur final reste toujours un problème. On a, d'une part, les utilisateurs qui ont des idées, mais qui n'ont pas l'environnement technique et les capacités en programmation pour faire eux-mêmes le développement. D'autre part, on a un grand volume des données, ressources et services qui qui pourraient être regroupées à la fois en termes de données, mais le plus important, en termes de comportement d'innover et de créer nouveaux objets.

Notre objectif dans cette thèse est de combler ce manque d'outils qui sont capables d'une interaction directe et l'autonomisation des utilisateurs finaux, de manière unifiée. Ainsi, notre principale contribution dans cette thèse est le développement d'une approche holistique pour les systèmes basés sur le Web qui sont centrés sur l'utilisateur et qui intègrent des données, les services et le comportement disponible sur le Web 2.0.

Mots clés :
développement web orienté utilisateur, mashups, services web, web 2.0, contexte, comportement

Abstract

World Wide Web (WWW) has become the greatest repository of information that man has ever assembled and it is continuously growing. WWW transformed itself into a generative environment that fosters innovation through the advance of technologies and a shift in people's perception of the Web and how they use it. The new WWW or Future Internet is that of an Internet of Services and Internet of Things.

Naturally, a series of questions arise from this context: how do you filter things to create more value than you currently get? how do you aggregate things in an intelligent and easy way instead of doing it in your head? The world cannot be described unambiguously, so how can you allow users to deal with the world in their own way, based on their understanding? Levine in his book "Cluetrain manifesto" was arguing that markets are conversations so how can users be involved in the conversation? how can users be empowered with easy consumption of the services, information, things that they found around?

However design and deployment of such software capable of direct interaction and empowerment of the end-user is still an issue. We have on one side users that have ideas, but do not have technical background and lack programming skills to do the development by themselves. On the other side, we have large amounts of data, resources and services that could be aggregated both in terms of data, but most important in terms of behavior to innovate and create new things.

Our goal in this thesis is to address this lack of tools that are capable of direct interaction and empowerment of end-users, in a unified manner. Thus our main contribution in this thesis is the development of a holistic approach for web based systems that are user-centric and that integrate data, services and behavior available on the Web 2.0.

Keywords :
web oriented end-user development, mashups, web services, web 2.0, context, behavior

Résumé étendu

World Wide Web (WWW) est devenu le plus grand dépôt d'informations que l'homme ait jamais assemblé et il est en croissance continue. WWW s'est transformé en un environnement génératif qui favorise l'innovation par le développement des technologies et par un changement dans la perception des gens sur le Web et comment l'utilisent [Zittrain, 2008]. Il "a passé des pages Internet basées sur la transaction aux celles basées sur l'interaction" [Ogrinz, 2009]. WWW a radicalement changé, aussi la façon dont les connaissances sont partagées, en abaissant la barrière pour la publication et l'accès aux documents [Bizer *et al.*, 2010]. En outre, par la prolifération de Web APIs, Web est devenu une plate-forme hautement programmable [Aghaee *et al.*, 2013]. Le nouveau WWW ou *l'Internet de l'Avenir* est celui d'un Internet des Services et un Internet des Objets.

Selon [O'Reilly, 2007], "Web 2.0 est la révolution des affaires dans l'industrie informatique causée par le passage à l'Internet comme une plate-forme, et une tentative de comprendre les règles de succès sur cette nouvelle plate-forme". Les technologies Web 2.0 sont interactives et obligent les utilisateurs à générer de nouvelles informations et contenus ou à modifier le travail des autres participants [Chul *et al.*, 2009]. Chul et al. continue à déclarer en [Chul *et al.*, 2009] que "la solution correcte vient des participants corrects".

Naturellement, une série des questions se posent à partir de ce contexte : comment filtrez-vous les objets pour créer plus de valeur que vous obtenez actuellement ? Comment pouvez-vous regrouper les objets d'une manière intelligente et facile au lieu de la faire dans votre tête? Le monde ne peut pas être décrit sans ambiguïté, alors comment pouvez-vous permettre aux utilisateurs de traiter avec le monde à leur manière, en fonction de leur compréhension? Levine dans son livre "Cluetrain manifesto" [Levine, 2009] a argumenté que les marchés sont conversations, alors comment peut-on impliquer les utilisateurs dans la conversation ? Comment peut-on autorise les utilisateurs à la consommation facile des services, informations, objets qu'ils trouvent autour d'eux ?

Cependant, la conception et le déploiement d'un tel logiciel capable d'interaction directe et l'autonomisation de l'utilisateur final reste toujours un problème. On a, d'une part, *les utilisateurs* qui ont des idées, mais qui n'ont pas l'environnement technique et les capacités en programmation pour faire eux-mêmes le développement. D'autre part, on a un grand volume *des données, ressources et services* qui qui pourraient être regroupées à la fois en termes de données, mais le plus important, en termes de comportement d'innover et de créer nouveaux objets.

Objective et contributions

Notre objectif dans cette thèse est de combler ce manque d'outils qui sont capables d'une interaction directe et l'autonomisation des utilisateurs finaux, d'une façon unifiée. Nous sommes préoccupés en particulier des outils fondés sur le Web. Nous affirmons que pour ces systèmes web centrés sur l'utilisateur, utilisateurs, services (dans la forme générale définie en [C. Lovelock, 1996]), sémantique

et contexte sont components clés.

Les mushups sont l'un des paradigmes du Web 2.0. Le terme *mashup* a été emprunté à la musique et représente une chanson ou une composition créée par le mélange de deux ou plusieurs chansons[1]. Dans le cas de web un mashup (voire par exemple [Altinel *et al.*, 2007]) a été défini principalement à partir d'une perspective technologique comme une application hybride qui emploi et combine de données, présentation ou fonctionnalité de deux ou plusieurs sources pour créer de nouveaux services, régulièrement par l'intermédiaire de Web APIs[2].

Le calendrier des conférences est qu'un exemple – parmi tant d'autres – qui exige un mélange d'idées (utilisateurs, services, données, comportement, contexte) discutés ici pour le rendre possible. Nous allons utiliser cet exemple comme un exemple récurrent, tout au long de cette thèse pour expliquer notre démarche.

Un tel calendrier est spécifique à l'utilisateur, puisque par exemple, certains utilisateurs pourraient être intéressés par web conférences liées, d'autres dans le web sémantique, d'autres règles ou de processus d'affaires des conférences. L'information contextuelle est mashé pour satisfaire l'objectif de certains utilisateurs, donc des informations spécifiques sur les conférences sont stockées dans un contexte de calendrier. Au moins deux services sont requis: celui qui traite avec les conférences et celui qui offre un calendrier. Du point de vue technologique ces services peuvent ne pas être compatibles les uns avec les autres.

Pour les scientifiques dans le domaine de l'informatique, la liste de diffusion DbWorld est l'endroit bien connu où ils peuvent rechercher une conférence en informatique. Une série d'informations sont fournies ici, mais le plus important c'est l'objet, le délai et la page Web de l'événement publié. D'un point de vue technique, DBWorld ne fournit pas d'API pour permettre l'accès programmatique et l'interrogation du service. En conséquence, par rapport à des approches de mashups actuelles, ce service est inutile.

D'autre part, le Calendrier Google est l'une des applications Google Apps les plus connus. L'information d'intérêt pour le calendrier est le titre de l'événement, la date et la description d'un événement. Ces informations se trouvent dans un Calendrier Google. Contrairement aux services DbWorld, Google fournit pour ce service, en plus de la représentation de la page Web, une API pour accéder au contenu. Cette situation est tout simplement un exemple qui soutient la nécessité d'être en mesure de traiter de façon non-uniforme l'accès aux services.

La voie habituelle pour atteindre cet objectif d'avoir des conférences stockées dans le Calendrier Google par leur date limite nécessite des interactions manuelles: (1) l'utilisateur doit maintenir deux onglets ouverts dans le navigateur; (2) même s'il peut y avoir plusieurs entrées qui sont conformes à un terme de recherche, l'utilisateur doit traiter les événements un par un parce que DBWorld ne fournit pas une construction dans la fonctionnalité de recherche; (3) l'utilisateur doit se déplacer entre les onglets ouverts à plusieurs reprises, afin de stocker un seul événement dans le calendrier, car un seul élément d'information peut être copié et collé à la fois (par exemple, le titre de l'événement).

Contribution

Le contenu de cette thèse réside à la confluence entre les thèmes suivants: Génie Logiciel, Architectures orientées vers les services, Sensibilité au contexte, Sémantique Web et Raisonnement, Management du procès des affaires et Systèmes d'information.

Le travail présenté tout au long de cette thèse est présenté principalement du point de vue du génie logiciel (Software Engineering). On concentre un peu sur la construction des théories et sur les rendre

[1] *http://en.wikipedia.org/wiki/Mashup_(music)*, retrieved 9 December 2013

[2] *http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)*, retrieved 9 December 2013

explicites [Sjøberg *et al.*, 2008, Easterbrook *et al.*, 2008] au sein du génie logiciel parce que nous employons des connaissances de plusieurs sujets, comme nous l'avons indiqué précédemment, nous trouvons important de préciser que la position philosophique [Easterbrook *et al.*, 2008], nous adoptons *le pragmatisme* pour notre méthode de recherche.

> Le pragmatisme reconnaît que toute connaissance est approximative et incomplète, et que sa valeur dépend des méthodes par lesquelles elle a été obtenue [Easterbrook *et al.*, 2008, Menand, 1997].

Pour les pragmatistes, la vérité est ce qui fonctionne à l'époque, entraînant un degré de relativisme. C'est-à-dire, ce qui est utile pour une personne peut ne pas être utile pour une autre, par conséquent la vérité est relative pour l'observateur [Easterbrook *et al.*, 2008]. Pragmatisme adopte une approche d'ingénierie pour rechercher la valeur des connaissances pratiques sur la connaissance abstraite. On utilise des méthodes de recherche mixtes pour faire la lumière sur la question à l'étude.

Notre contribution principale est le développement d'une approche holistique pour les systèmes web qui sont centrés sur l'utilisateur et qui intègrent les données, les services et le comportement disponible sur le Web 2.0.

Nous croyons que l'approche que nous avons développée nous amène un pas de plus près pour permettre aux utilisateurs finaux de programmer leurs propres applications. Par ailleurs l'approche que nous avons développée est de compléter les techniques existantes.

En détails, nos contributions sont présentées en plusieurs étapes:

- **Étape 1ère :** Nous commençons par aborder l'aspect conceptuel du problème. Nous revisitons les technologies web actuelles avec un accent sur les applications mashups à partir d'un point de vue conceptuel et de discuter les concepts que nous allons utiliser dans notre approche.
- **Étape 2ème:** Nous proposons une architecture qui hérite des caractéristiques de l'appareil de la métaphore TomTom, et considère l'utilisateur comme étant un composant du système. Les principes de la SOA, SaaS et Web sont mélangés dans une solution hybride.
- **Étape 3ème:** Nous développons la sémantique opérationnelle du système.
- **Étape 4ème:** Nous discutons l'implémentation de la mise en œuvre de tels systèmes, nous fournissons un ensemble de cas d'utilisation, et nous décrivons un prototype roulant qui constitue une preuve de concept de notre approche.

Chaque étape est présentée dans un chapitre dédié.

Chapitre 2 est consacré à l'examen des travaux connexes. Des différentes approches Mashups sont discutées avec les tendances influentes et les technologies qui forment les bases pour le développement des mashups. Nous discutons aussi la raison pour laquelle nous considérons mashups que des travaux connexes pertinents et nous exposons notre opinion sur la raison pour laquelle certaines des techniques de mashups ont été abandonnées ou ont été intégrées dans les grands projets. Nous concluons le chapitre avec une liste des exigences auxquelles l'approche et le système que nous allons présenter dans les prochains chapitres doivent se conformer.

Chapitre 3. Dans ce chapitre nous présentons notre approche conceptuelle. Cette approche est fortement orientée vers l'utilisateur final. L'approche prévoit un système composite qui comprend un utilisateur humain et un système intelligent. Nous discutons des aspects conceptuels qui animent notre approche: l'utilisateur final / centrée sur l'utilisateur; plan; système à deux couches; système intelligent;

humaine - l'interaction du système. Deuxième partie du chapitre décrit le modèle conceptuel associé à notre approche.

Chapitre 4. Nous proposons une architecture pour les systèmes que nous discutons tout au long de cette thèse. Cette architecture suivie la métaphore TomTom. Elle est influencée par Newell [Newell, 1994] Model du processeur humaine et hérité de l'architecture orientée vers les services et de l'architecture orientée vers la distribution.

Chapitre 5. Nous proposons un modèle d'exécution basé sur une technique de raisonnement de chainage avant.

Chapitre 6. Ce chapitre affirme une série de lignes directrices de mise en œuvre sur la façon dont ces systèmes devraient être mis en œuvre. Une série de cas d'utilisation sont présentés dans ce chapitre. Ces cases d'utilisation seront employées pour valider notre approche. Nous discutons également de la mesure dans laquelle les exigences que nous avons identifiées dans le chapitre 3 ont été respectées. Un prototype roulant de notre approche sera décrite dans ce chapitre.

Chapitre 7. Les conclusions du travail présenté dans cette thèse sont énumérées dans ce chapitre avec les futures étapes sur façon dont ce travail peut être amélioré.

Chapitre 2 Résumé - Travaux connexes sur Mashups

Dans le deuxième chapitre, nous examinons les travaux existants dans le domaine des services Web, Web 2.0, Web sémantique, les services Web sémantiques, Web des données présentant leurs influences sur le développement d'outils de mashup.

Au meilleur de nos connaissances, nous ne sommes pas au courant de toute autre enquête sur ces dimensions et qui traite de tels nombreux aspects liés au développement de mashups.

Mashups sont l'une des paradigmes Web 2.0 et une zone de l'application End User Development (EUD) prometteuse [Grammel et Storey, 2008]. Nous croyons qu'ils sont le plus étroitement liés du type du case d'utilisation que nous avons énoncé dans le Chapitre ch:introduction.

Plusieurs définitions ont été données pour définir le concept de *mashups*. Altinel et al. Altinel et al. [Altinel *et al.*, 2007] définit un mashup comme

> une application web qui combine le contenu de deux ou plusieurs applications pour créer une nouvelle application. Les applications situationnelles sont des applications web d'entreprise reposant sur la volée pour résoudre un problème commercial spécifique. Elles sont souvent élaborées sans la participation du département informatique et opèrent en dehors de son contrôle. Elles combinent les données provenant d'une variété de sources d'entreprise telles que SAP ou des applications Office, bases de données back-end, et les systèmes de gestion de contenu.

Bien que toutes les définitions fournies soulignent plus ou moins les mêmes caractéristiques, Eric Schmidt de Google définit ces nouvelles applications comme

> "applications qu'on l'assemble " - avec les caractéristiques que les applications sont relativement faibles, les données sont dans le nuage, les applications peuvent fonctionner sur n'importe quel appareil (PC ou mobile), les applications sont très rapides et très personnalisable, et sont réparties de manière virale (réseaux sociaux, e-mail, etc).[3]

[3] http://www.youtube.com/watch?v=T0QJmmdw3b0, retrieved 17 December 2013

Un aspect que nous soulignons dans la première partie de notre étude est que la plupart des services Web (par exemple SOA, Web sémantique et le Web des données, Software as a Service) les technologies relatives ont influencé d'une certaine manière le développement de mashups. Nous continuons plus tard avec la présentation d'une large palette de mashups approches: mashups comme une collection de widgets, mashups basés sur les tuyaux; des approches de mashups hybrides; domaine des langues spécifiques pour les mashups.

Sur la base des nombreuses réflexions sur les tendances qui ont influencé la création de mashups ainsi que sur la base des orientations de recherche identifiés par le Groupe d'experts FP8 l'UE travaillant sur les services dans l'Internet du futur [4], dans les paragraphes qui suivent nous allons faire une série d'affirmations et considérations sur les raisons pour lesquelles nous croyons que les approches actuelles n'ont pas vraiment atteint le résultat souhaité ; nous extrayons un modèle commun minimal pour mashups basé sur les approches de mashups présentées dans le chapitre ; nous énumérons les exigences que nous considérons avec lesquels une approche de mashup doit se conformer. Notre approche, que nous allons élaborer et présenter dans cette thèse sera conforme aux besoins identifiés.

Bien que les mashups ont été d'abord pensés et développés qu'à partir d'un point de vue technique, principalement par le biais des hacks, nous croyons que le côté conceptuel du problème doit être pris en compte.

Le développement de mashup représente une zone d'application du End User Development (EDU) prometteuse comme le fait valoir dans [Grammel et Storey, 2008]. En utilisant les services qui peuvent être accessibles par le Web comme plate-forme sous-jacente, l'effort de développement est passé de la programmation traditionnelle. C'est pourquoi les difficultés rencontrées par les designers des outils mashup comprennent la nécessité de définir un moyen de haut niveau pour la description et la combinaison du calcul, de la logique d'intégration et des abstractions pour représenter les widgets Web, les services Web, les sources de données sur le Web [Aghaee *et al.*, 2012]. Néanmoins, ces objectifs n'ont pas encore été atteints.

Un grand nombre d'outils de mashups ont été développés à la fois dans les universités et dans l'industrie, mais seulement quelques-uns ont réussi. Un grand nombre de ces approches, même si ont été développés par les grandes entreprises comme Microsoft ou Google ont été abandonnées, à savoir Microsoft Popfly, Google Mashup Editor. D'autres, comme JackBe Presto ou Serena Mashup Composer sont encore en usage. Nous croyons qu'il y a un grand intérêt pour ces outils tant dans le milieu universitaire que dans l'industrie et de nouveaux cadres sont en cours de développement, c'est à dire [Aghaee *et al.*, 2013]. Dans le même temps, nous pensons que les approches actuelles n'ont pas vraiment atteint le résultat souhaité car elles étaient soit trop technique ou trop scientifique, et qu'elles n'avaient pas une bonne intégration entre la perspective technique et la perspective conceptuelle. De cette manière, ces approches ont perdu sur le chemin la cible fondamentale - l'utilisateur final. Nous croyons que cet état vient du fait qu'à l'heure actuelle le processus de développement de logiciels en génie logiciel est principalement axée autour de la structure du logiciel en cours de construction et des interactions entre ses composants. Alors qu'en réalité l'accent devrait être mis sur les utilisateurs finaux.

Les clients de mashups Albeit qu'on fait valoir dans [Bioernstad et Pautasso, 2007] sont mashup réels, la quasi-totalité des approches que nous avons discuté dans ce chapitre sont les mashups côté du serveur. Nous croyons que les approches centrées sur l'utilisateur pour les mashups devraient être des mashups basés sur le client. Les utilisateurs finaux ne devraient pas être tenus d'installer, de configurer et de maintenir des serveurs. Les utilisateurs finaux doivent toutefois recevoir les outils qui sont en

[4] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, extrait le 13 janvier 2014
http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, extrait le 13 janvier 2014

général suffisantes pour répondre d'une manière unifiée et holistique à la plupart des technologies énumérés ici. Ces outils doivent cacher autant que possible le côté de l'ingénierie et les technologies. En outre, ces outils devraient être les mêmes, peu importe le type d'appareil qui est utilisé. Nous affirmons que ces outils peuvent être atteints par l'avance de HTML5 et JavaScript. Nous pensons qu'une solution basée sur un navigateur peut se conformer à tous ces aspects. D'autres chercheurs sont d'accord avec nous sur ce sujet. On pourrait voir par exemple [Aghaee et Pautasso, 2010].

La liste des exigences que nous croyons un système de mashup orientée vers l'utilisateur final doit respecter :

E1 un tel système doit permettre l'évolution, le partage et la distribution ; les utilisateurs finaux doivent être autorisés par saisie directe de mettre à jour / adapter l'application;

E2 un tel système ne devrait pas être un domaine spécifique, et devrait permettre un large éventail de cas d'utilisation;

E3 dans un tel système l'utilisateur final devrait être le coordonnateur de la façon dont le système fonctionne, d'où le système doit permettre un nouveau modèle de programmation pour les systèmes composites (homme + services) [5];

E4 un tel système devrait soutenir les développeurs qualifiés ainsi que les utilisateurs novices;

E5 un tel système devrait se concentré sur l'utilisateur final et pas sur le système lui-même. Le système doit être caché à l'utilisateur final autant que possible en fournissant le bon niveau de représentation tels que la représentation de problème pourrait être traduit automatiquement dans les concepts de base de la langue de programmation sous-jacent dans lequel le système global est mis en œuvre;

E6 un tel système devrait permettre sur le développement de la demande en utilisant le web comme une plate-forme Web comme un environnement d'exécution de l'application: s'appuyer sur des idées, des sites, des applications existantes [6];

E7 un tel système doit être conforme aux principes de la SOA de: couplage de poux, réutilisation, possibilité de découvrir, compossibilité;

E8 un tel système doit permettre l'exécution décentralisée et délocalisée de logiciels / composants [7];

E9 ces systèmes devraient permettre un moment de la construction simultanée, le développement des temps d'exécution et l'expérience [8].

Ayant cette discussion comme point de départ, nous présenterons dans le chapitre 3 de notre approche ainsi que le modèle conceptuel que nous proposons d'aborder les questions soulignées dans le chapitre 2.

[5] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, extrait le 13 janvier 2014
[6] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, extrait le 13 janvier 2014
[7] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, extrait le 13 janvier 2014
[8] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, extrait le 13 janvier 2014

Chapitre 3 Résumé – Une approche centrée sur l'utilisateur. Model conceptuel

Nous avons souligné dans le chapitre 2 une série de caractéristiques et des exigences avec lesquels nous croyons que les mashups centrés dur l'utilisateur doivent se conformer.

Ainsi, tels systèmes devraient être centrés sur l'utilisateur. Tels systèmes devraient permettre un nouveau modèle de programmation pour les systèmes composites (hommes + services) E3.. Nous soutenons que ces systèmes soient des systèmes intelligents étant capable d'interagir avec l'utilisateur final selon un plan convenu à l'avance, en soutenant l'évolution, le partage et la distribution E1. Par conséquent, ces systèmes sont deux systèmes de couches: une couche de haut niveau, qui traite le problème à un niveau conceptuel et sémantique (l'accord sur le plan) et une couche de bas niveau qui traite des internes du système et des technologies de bas niveau, par exemple, accès direct aux services, etc. La couche de bas niveau doit être cachée, autant que possible de l'utilisateur final E5.

Les aspects (également représentées sur la figure 1) qui conduisent au développement de notre approche sont: orienté vers l'utilisateur final ou centrée sur l'utilisateur ; les humains et le système d'interagir avec l'autre; régime; système à deux couches; système intelligent. Ces aspects ont déjà été discutés dans la littérature de recherche, mais presque toujours d'une manière de déconnexion avec presque pas d'interaction entre eux. En outre une terminologie différente, selon les orientations de la recherche où il a été étudié, a été utilisée pour identifier réellement le même concept.

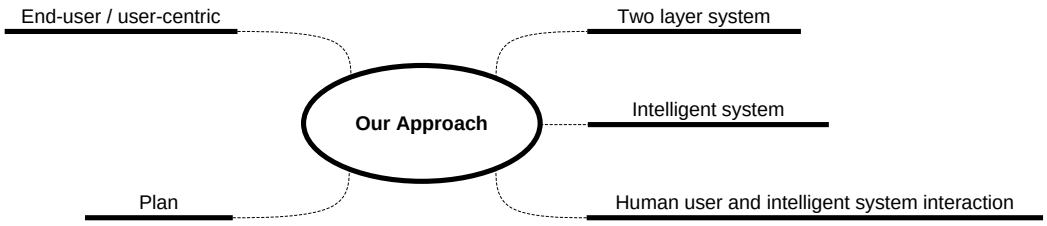


Figure 1: Aspects that drive our approach

Ce chapitre décrit l'approche proposée par nous et le modèle conceptuel que nous avons crée dans la relation avec notre approche.

L'approche

Le développement de l'utilisateur final a été défini dans [Lieberman *et al.*, 2006] comme une

> une série des méthodes, techniques et outils permettant aux utilisateurs des systèmes logiciel qui agissent comme développeurs de logiciel non-professionnel, à un certain point de créer, modifier ou étendre un artefact logiciel.

La recherche en génie logiciel de l'utilisateur final est interdisciplinaire [Lieberman *et al.*, 2006] impliquant les idées : Informatique, génie logiciel, l'interaction homme-ordinateur, de l'éducation, de la psychologie et d'autres disciplines.

Les *utilisateurs finaux* que nous voulons soutenir, sont la plupart, *pas des développeurs professionnels*, qui manquent de compétences techniques, et qui n'ont pas les connaissances nécessaires pour écrire des programmes logiciels, selon les spécifications techniques (cahier des charges pour les

services Web, les API, REST etc.). Les utilisateurs finaux ont une variété des *objectifs*. Ces objectifs sont atteints par *la création* de nouvelles applications, par brassage des applications existantes, ou des artefacts logiciels, par *la modification* ou *l'adaptation* des applications existantes.

Maintenant, afin de permettre aux utilisateurs finaux, qui ne sont pas des programmeurs professionnels à programmer, adapter les applications existantes ou des artefacts logiciels en fonction de leurs besoins la couche technique a besoin d'être caché, autant que possible, sinon ils ne seront pas en mesure de le faire.

Pour réaliser cette séparation des préoccupations et ainsi cacher la couche technique, nous soutenons qu'il est nécessaire ***un système à deux couches***. La couche de haut niveau devrait fournir les moyens pour permettre à deux utilisateurs de l'homme et le système de comprendre l'autre en utilisant un ensemble commun de concepts et suite à une avance d'accord sur ***le plan***. D'autre part, la couche de bas niveau doit être caché à l'utilisateur final et doit être accessible directement par le système en fonction de ce qui a été convenu dans ***le plan***. Un développeur professionnel devrait également être autorisé à accéder à cette couche.

Le génie logiciel est le domaine de l'étude qui se préoccupe de tous les aspects liés à la conception et le développement de systèmes logiciels. Deux de ces aspects: ingénierie des exigences et la conception du système sont d'intérêt pour notre approche, parce que le comportement interne du système cible alors que les besoins sont externes, concernant le monde.

La phase de la collecte des besoins précède la conception du système. Malheureusement, dans la plupart des cas, le produit final n'est pas conforme aux attentes de l'utilisateur final pour diverses raisons: à savoir une mauvaise communication, une compréhension différente des concepts et des situations, etc. [Tognazzini, 1992, Nardi, 1993]. Un utilisateur final perçoit et comprend un système logiciel par l'intermédiaire de l'interface d'utilisateur (UI). Basé sur l'interface d'utilisateur, qui est devrait être une représentation exacte et complète du système, l'utilisateur final construit sa propre compréhension de l'environnement constitué de concepts avec lesquels il travaille, et le comportement associé [Clark et Sasse, 1997]. Impuissante dans de nombreuses situations de la compréhension de l'utilisateur final est très différent de la compréhension et le message, les développeurs ont essayé effectivement de transmettre [Tognazzini, 1992].

Une exigence dans l'ingénierie des exigences (Requirements Engineering (RE)), comme indiqué dans [Pohl, 2010],

> définit tant les besoins que les objectifs des utilisateurs, et les conditions et propriétés du système à développer, ce résultat, par exemple, de besoins organisationnels, des lois ou des normes.

Dans l'ingénierie des exigences un objectif est l'intention de parties concernant les objectifs, les propriétés ou l'utilisation du système [Pohl, 2010]. Les exigences définissent ce qui doit être développé tandis que la conception de système définit la façon dont le système doit être développé [Pohl, 2010].

En conséquence, nous débattons que l'ingénierie des exigences signifie la couche de haut niveau tandis que la conception du système correspond à la couche de bas niveau. Et par conséquent, nous croyons que les deux l'ingénierie des exigences et la conception du système doivent être unifiés lorsqu'il s'agit de systèmes intelligents appropriés centrés sur l'utilisateur (voir figure 2).

Les approches de l'ingénierie des exigences proposent aussi l'utilisation de scénarios, qui représentent des exemples concrets, positifs ou négatifs de satisfaction ou l'échec de satisfaire un objectif ou un ensemble d'objectifs [Pohl, 2010]. Tels scénarios pour notre approche sont ***les plans*** que les utilisateurs finaux créent. Configuration logicielle sont généralement exprimés en langage naturel. Cependant le langage naturel ne peut pas être une option ici. Nous avons besoin d'un ensemble structuré et bien défini de concepts pour exprimer les plans de l'utilisateur final, telles que un système intelligent peut

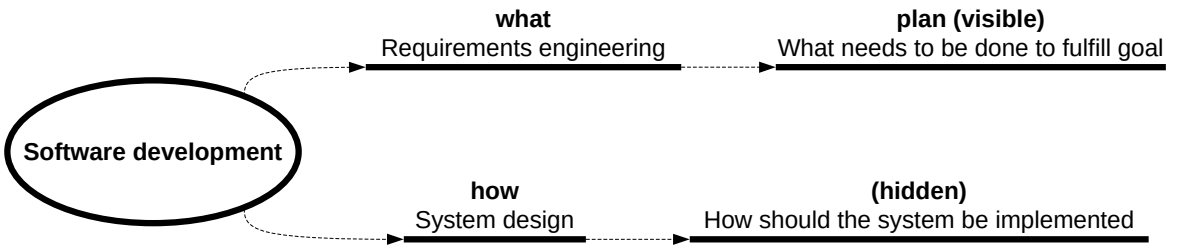


Figure 2: Software development aspects

comprendre, raisonner et utiliser ce plan, dans son interaction avec l'environnement et l'utilisateur final.

Par conséquent, un plan d'utilisateur final comprend un ensemble de concepts, avec laquelle l'utilisateur final fonctionne afin de satisfaire un objectif, leur relation les uns avec les autres, le cadre, et un moyen pour exprimer le comportement aux formes de processus et des règles. Ainsi, pour notre approche l'utilisateur dispose d'un plan de ce qui doit être fait *pour atteindre un objectif. Les besoins à faire* ici signifie *l'interaction (comportement)* que l'utilisateur doit exposer avec l'application (applications) afin de remplir l'objectif. En outre l'utilisateur attend une réponse particulière du système en réponse à des actions il / elle, l'utilisateur, effectue. Le comportement de l'utilisateur est imposé (influencé) par le contexte *(environnement)*.

Pour l'approche que nous envisageons et conceptions dans cette thèse, l'utilisateur crée un *plan* qui est partagé (donné) au système. Ce plan contient une description du *contexte(s)* et *le comportement* que tant l'utilisateur humain et le système doivent effectuer en relation avec le contexte (s) comme nous avons introduit dans [Pascalau, 2011a]. Le plan explique comment le système doit réagir en réponse aux actions que l'utilisateur humain effectue, ou comment le système doit réagir en réponse aux changements qui apparaissent dans l'environnement (contexte(s)). De cette façon, tant l'utilisateur humain que le système suivront et partagent la même compréhension, le même plan.

Les deux derniers aspects que nous avons identifiés concernant notre approche sont les suivants: *système intelligent* capable *d'interagir avec l'utilisateur humain selon le plan convenu*.

Nous affirmons que notre système peut être assimilé à la notion d'un agent. Un agent hybride: une combinaison entre un agent réactif, un agent déductive et également agent proactif.

Nous relions ainsi les aspects que nous venons de discuter et nous reprenons notre approche à la figure 3. L'approche que nous avons introduite dans cette section propose un nouveau modèle de programmation grâce à un système composite où l'utilisateur humain et le système se trouvent en interaction les uns avec les autres. L'interaction se fait via l'environnement et selon un plan prédéfini. Ce plan est créé par l'utilisateur humain, d'où l'homme est le coordinateur de la façon dont le système fonctionne. Tant le système intelligent que l'utilisateur humain suivent le même plan. Ce plan sert à atteindre un objectif particulier et elle a été créée prenant en compte le contexte(s). Interaction est perçue par les changements qui apparaissent dans l'environnement.

Le modèle conceptuel

Nous avons annoncé précédemment que la manière appropriée de définir notre modèle conceptuel est au moyen d'ontologies [Guarino, 1998]. Les instances de ce modèle représenteront *le plan* que l'utilisateur va fournir au système.

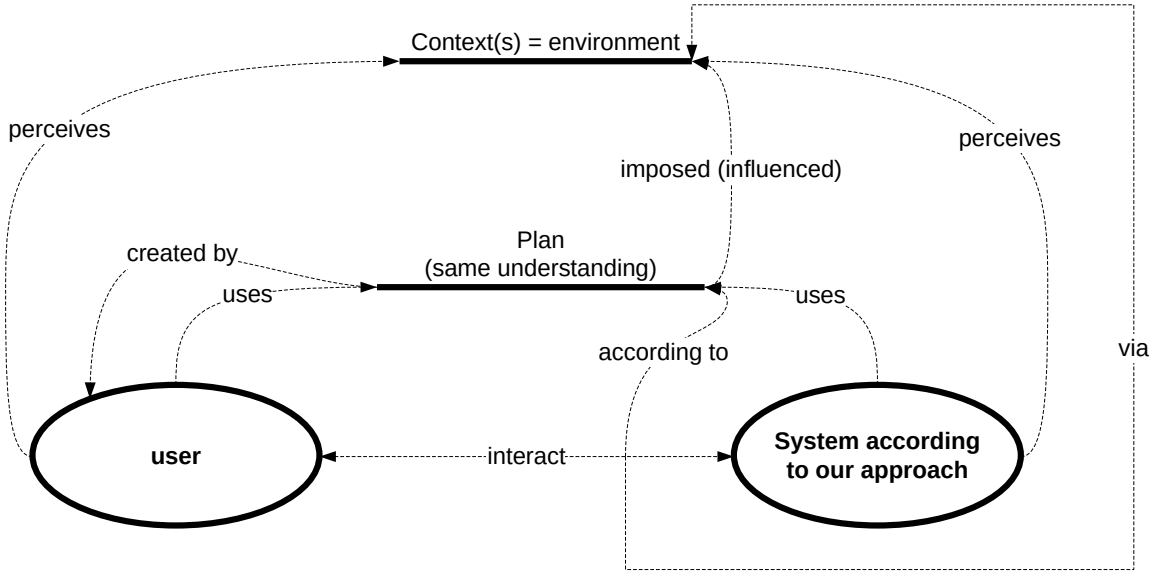


Figure 3: Our approach

Comme le UML est considéré *le standard* modelant le langage [Guizzardi, 2005], nous l'employons aussi pour formaliser notre cadre conceptuel.

Concept

Bien que de nombreuses approches ontologiques (voir, par exemple, OWL [Group, 2009]) utilisent comme l'entité de niveau supérieur, la notion d'objet pour notre cadre conceptuel l'unité la plus générale de la connaissance est la notion de concept. En outre, la spécification de l'OMG pour la Sémantique du Vocabulaire des Affaires et les Règles (SBVR) [OMG, 2008] utilise comme entité de top la notion de *concept* notion.

Un concept a un nom et un ensemble de propriétés. Il s'agit d'une sous-classe `uml::Class` entité. Il peut être identifié soit par son nom ou par l'ensemble (ou un sous-ensemble) des propriétés qui le définissent. En génie logiciel lorsqu'il s'agit de langages typés, les entités sont reconnues par leurs types (nom de la classe). Le sens inverse est basé sur un ensemble de caractéristiques.

Dans notre approche les deux points de vue sont pris en compte.

Un concept représente une unité de connaissance créée par une combinaison unique de caractéristiques. Ainsi, comme le montre la figure 4, chaque élément du cadre est un concept. De cette façon, le processus de raisonnement peut entraîner aucune des concepts définis de manière unifiée.

Rappelons le cas d'utilisation du calendrier des conférences que nous avons introduit dans le chapitre 1. L'objectif est de stocker automatiquement les événements qui sont annoncés dans la liste de diffusion DbWorld dans un calendrier Google. Sans une approche similaire à celle décrite dans cette thèse, pour atteindre cet objectif de stocker automatiquement les événements dans un calendrier Google on a besoin de deux onglets ouverts dans le navigateur, nécessaire pour aller et retour entre ces deux onglets et copier et coller chaque élément d'information manuellement pour chaque événement. La figure Figure 5 représente le service de calendrier Google - *l'événement facette de sauvegarde*.

Le service de calendrier Google est un exemple de concept de `Service`. Depuis un concept possède des caractéristiques, la caractéristique d'intérêt pour nous est l'`URL` du service:

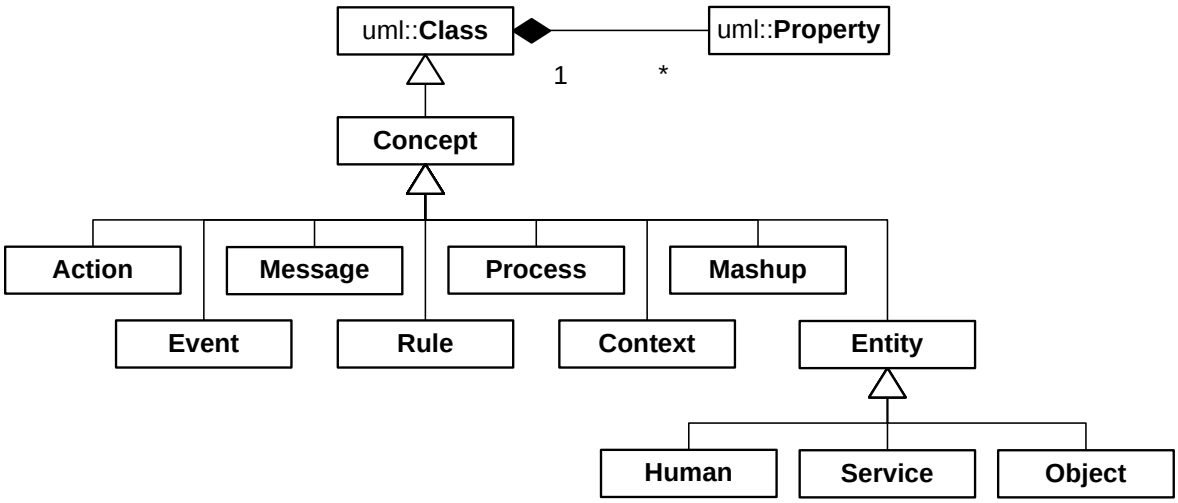


Figure 4: Concepts

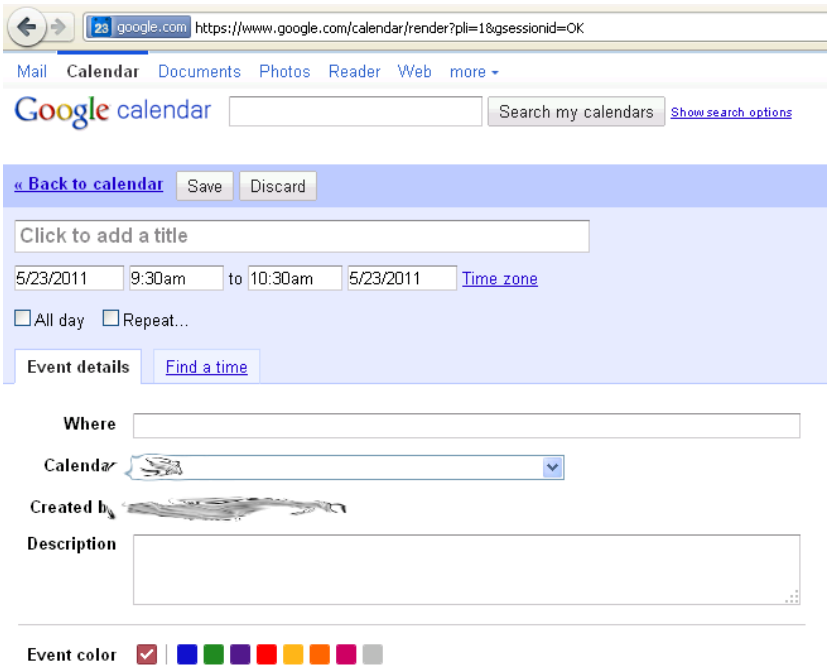


Figure 5: Google calendar service - save event facet

https://www.google.com/calendar/. Un autre concept de l'intérêt pour notre cas d'utilisation est l'existence du bouton `Save`. Empiriquement à partir d'un point de vue de l'utilisateur final d'identifier le bouton Sauver comme le bouton de sauvegarde exige que le bouton est un `bouton` et a le contenu du texte `Save`. Une représentation réelle, comme un arbre DOM, du bouton d'enregistrement selon le modèle de Google actuelle, à l'intérieur d'un navigateur, est représenté dans l'exemple 0.0.1. Ainsi, dans le but d'identifier le bouton Sauver, la représentation de l'arbre DOM du service de calendrier Google doit contenir un concept qui présente les caractéristiques suivantes: `type` de valeur `div`, `class` avec la valeur `goog-imageless-button-content`, et `textContent` avec la valeur

`Save`.

Example 0.0.1 (Google Calendar Save Button DOM representation)**.**

```
<div class="goog-imageless-button-content">
Save
</div>
```

D'une façon similaire on peut identifier tous les autres concepts nécessaires pour ce cas d'utilisation.

Contexte

Pour notre cadre conceptuel la notion de contexte adhère à l'appareil mathématique défini dans [Analyti *et al.*, 2007], mais ici le contexte est un ensemble de concepts et pas un objet.

Un concept constitué d'un identificateur de contexte et un ensemble d'identificateurs de concepts.

Basé sur le mécanisme d'identification des concepts le contexte est identifié respectivement en identifiant de manière récursive tous les concepts constitutifs.

Example 0.0.2. Un contexte dans notre cas d'utilisation Calendrier Google comprendrait, par exemple, le concept se référant au service de calendrier Google et les deux concepts nécessaires pour identifier le bouton Sauver. Ainsi, le système afin d'être en mesure de dire qu'il est en cours d'exécution dans ce contexte particulier on doit trouver les trois concepts qui sont compris dans ce contexte.

Comportement

Traditionnellement le comportement a été défini par des *business rules* et des *business processes* [Weske, 2007]. Notre cadre conceptuel est d'accord avec cette approche. En outre, nous pensons que le comportement est fortement lié au contexte(s) [Pascalau, 2011a]..

Le comportement d'une entité représente l'ensemble des événements, des actions et des messages que cette entité produit. Un événement est une occurrence d'un phénomène observable. C'est quelque chose qui "se passe", un événement qui est détectée, soit un `click` sur un bouton. Un événement a la même signification tant dans le cas d'une règle ainsi que d'un processus. Les événements peuvent être connectés à la fois.

Le comportement est un ensemble de règles et / ou un ensemble de processus, ou une combinaison des deux, liée au contexte(s).

Example 0.0.3 (Exemple de Règle)**.**

```
Si un événement de clic a été relevé de la touche de recherche réside dans le contexte de
services de calendrier et il y a un champ de recherche dans ce même contexte et la valeur
du champ de recherche est trouvée dans la colonne de l'objet du contexte de service
DbWorld et il y a pour chaque événement une date de départ dans la même ligne que la
 colonne de l'objet et un emplacement, puis sauver cette entrée d'événement.
```

Mashup

En conséquence un mashup est un plan (carte) qui décrit le contexte(s) et le comportement afférent qu'un utilisateur doit faire pour atteindre un but désiré. Une telle mashup est définie à partir d'un point de vue de l'utilisateur.

Un mashup est un ensemble de contextes et comportement.

La figure 6 illustre le cadre général. Ainsi, le concept de Mashup contient une ou plusieurs contextes. En outre, un Mashup peut contenir processus, règles ou une combinaison des deux. Un contexte est essentiellement une collection de concepts. En outre, un contexte pourrait avoir des sous-contextes. Un contexte fait référence à une entité.

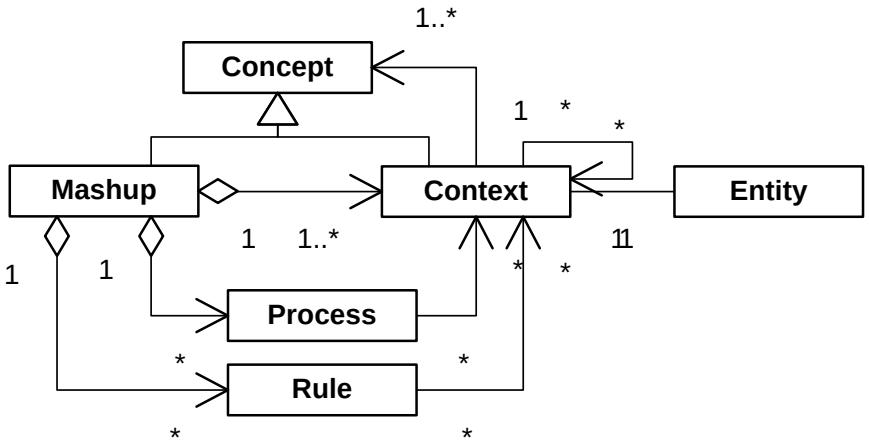


Figure 6: Mashup Concept

Chapitre 4 Résumé – Architecture

L'architecture que nous proposons dans ce chapitre suit l'approche que nous venons d'introduire auparavant. Au meilleur de notre connaissance, il n'existe pas de système liés au contexte web: (1) qui suit une approche à deux couches pour la conception du système, (2) qui utilise une représentation unique étant compris tant par l'utilisateur que par le système de la même voie; (3) qui estime et exige à l'utilisateur final de participer activement dans le système afin d'atteindre un objectif désiré.

En dépit de cette lacune dans le contexte Web afférent - dont nous soutenons qu'elle peut être remplie avec une approche similaire à celle introduite dans cette thèse – il y a une approche similaire dans les caractéristiques au sein d'un domaine différent. Le système TomTom[9] est probablement l'un des plus connu et le plus évolué [TomTom, 2007]] parmi les systèmes global de positionnement (GPS)[10]. Ainsi TomTom est un exemple de la métaphore du GPS que nous l'avons précisé dans l'introduction de cette thèse. L'architecture que nous proposons hérite des appareils TomTom.

Dans la figure 7, est représentée la métaphore GPS d'un système complet nécessaire pour trouver une personne, par exemple, de Paris à Berlin. Les acteurs impliqués sont l'utilisateur humain (qui est au volant d'une voiture) et le système intelligent qui dans ce cas est le système GPS. Tant l'utilisateur humain que le GPS utilisent un plan (ici une carte) pour atteindre l'objectif souhaité de se rendre à Berlin. La carte contient la description du contexte(s) à savoir les villes, les routes, etc. Le comportement est également spécifié. Une particularité de ce système est que pour définir le comportement, l'utilisateur introduit directement ce comportement en spécifiant la route exacte à suivre et dans ce cas le système GPS vérifie si l'utilisateur humain a dévié du comportement défini. Ou bien, la deuxième possibilité consiste simplement à définir le début et la fin de l'emplacement et le GPS calcule l'itinéraire qui doit être suivi par l'utilisateur humain. L'utilisateur final est également une partie du système, parce que si l'utilisateur ne conduit pas le véhicule, le but recherché ne sera jamais atteinte. Au moment même où le système GPS perçoit l'environnement par la réception des événements par les satellites il est en mesure de calculer en permanence la position actuelle ainsi.

Nous croyons que les similarités entre notre approche et la métaphore du GPS peuvent être facilement observés. La métaphore GPS suive en même temps le système à deux couches que nous

[9] http://www.tomtom.com/
[10] http://en.wikipedia.org/wiki/GPS

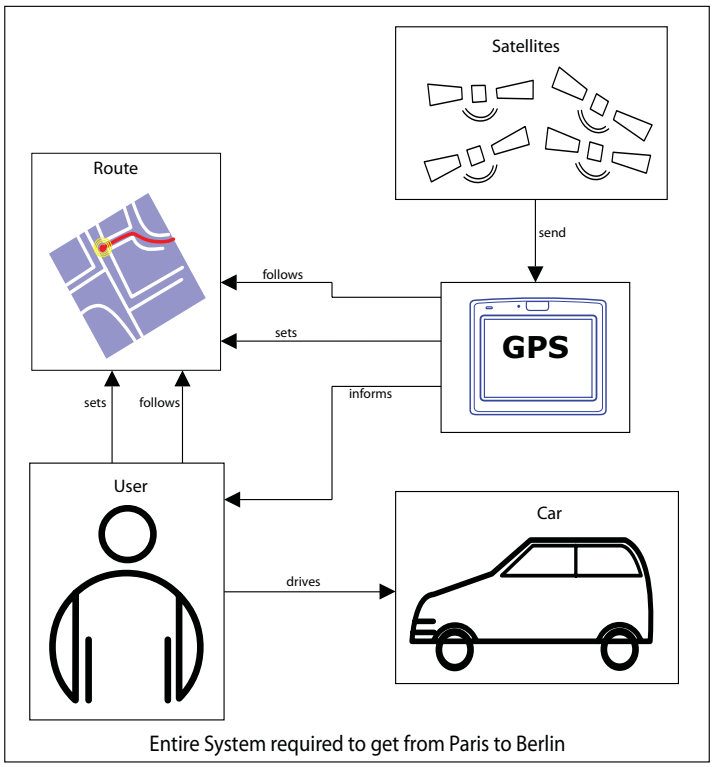


Figure 7: GPS Example

avons identifié pour notre approche. De même, on utilise un plan qui a été défini par l'utilisateur final et puis donné eu système intelligent. Le plan (la carte) dans le cas du GPS comprend aussi une définition du contexte(s) et le comportement qui doit être suivi tant par l'utilisateur humain que par le système intelligent. Le plan est compris de la même façon par toutes les parties concernées. Le comportement est perçu aussi par les changements qui apparaissent au sein de l'environnement (contextes). Tant l'utilisateur humain que le système sont nécessaires, comme components actives du système général, pour attendre l'objectif désiré par l'utilisateur final. Par conséquent, nous soutenons que la métaphore du GPS est un exemple précis de notre approche.

De notre point de vue les aspects les plus importants que nous apprenons de l'appareil TomTom sont: d'abord que, la couche de bas niveau du système (le niveau du système) a été conçue par défaut comme système en temps réel et la seconde qu'au niveau du système, il y a une façon unifiée pour représenter l'information. Nous soutenons que ces deux aspects constituent le sous-sol et sont des exigences fondamentales pour les systèmes conformes à la démarche que nous avons introduit la construction. Etre en temps réel permet la spécification du comportement. La manière unifiée pour la représentation de l'information est l'obligation pour le plan défini par l'utilisateur final et pour la construction d'un système intelligent qui peut être conscient de lui-même et de l'environnement.

L'architecture

L'architecture que nous présentons ici concerne les navigateurs web, en raison de plusieurs raisons. Tout d'abord, selon les derniers commentaires qui ont fini la section précédente, les navigateurs Web sont en effet en temps réel, et ils offrent aussi une façon unifiée pour représenter l'information. En plus

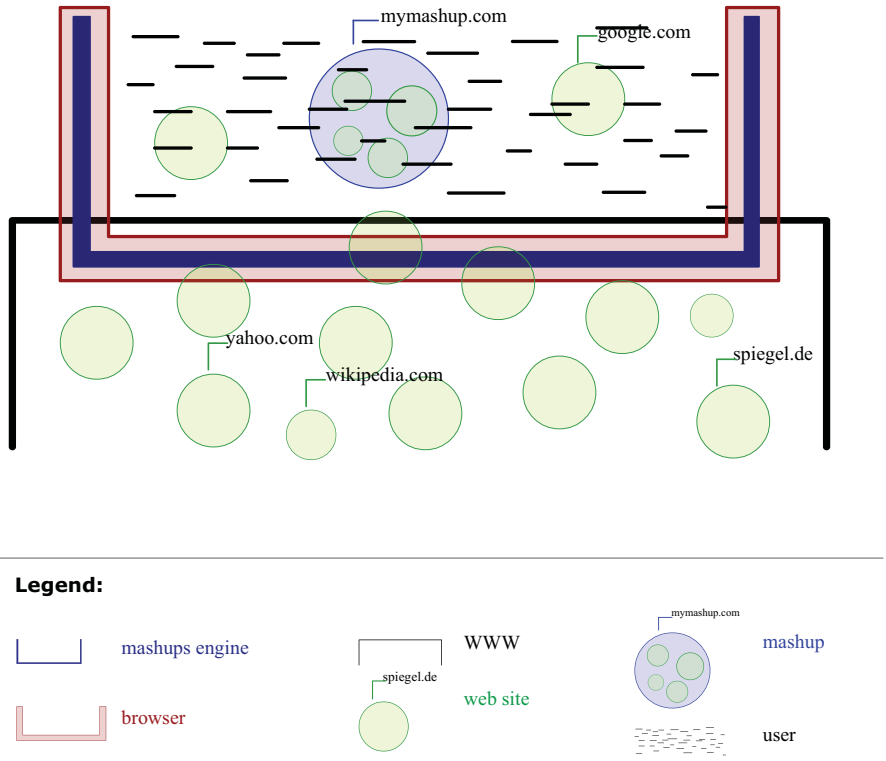


Figure 8: The General Architecture

de la représentation unifiée de l'information, les navigateurs Web fournissent également une manière unifiée de programmation et accès. En outre, presque tous les utilisateurs finaux savent comment utiliser un navigateur Web ; un navigateur web existe presque sur tous les dispositifs réels, allant de consoles de jeux, les récepteurs de télévision, les téléphones intelligents, les tablettes, les ordinateurs de bureau. La technologie est la même partout ; toutes les systèmes d'opération ont été construites comme un navigateur web, voir par exemple Google Chrome OS[11], Firefox OS[12]; un grand nombre de kits de développement du logiciel pour la création d'applications mobiles multiplateformes sont construites en utilisant HTML5 + JavaScript. Voir par exemple Sencha Touch 2[13], PhoneGap[14], jQuery Mobile[15].

L'architecture générale d'un système respectant l'approche que nous avons introduit (voir la Figure 3) et qui est similaire aussi avec les appareils TomTom sont illustrés dans la Figure 8.

Cette architecture est adaptée pour le web. Comme on peut le voir de l'image, le moteur de mashup fait partie du navigateur, ce qui lui donne un accès privilégié tant au contenu du navigateur lui-même qu'aux pages Web qui peuvent être accessibles via le navigateur.

On peut imaginer tout le système comme un type spécial d'aquarium qui n'a pas de fond. Le navigateur enrichi d'un tel moteur de mashup est enfoncée dans l'eau. Ainsi, l'utilisateur du navigateur et par conséquent l'utilisateur du moteur de mashup ont accès à tous les services auxquels le navigateur et le moteur de mashup ont accès. Tant l'utilisateur du navigateur que le moteur de mashup "regardent"

[11] http://www.chromium.org/chromium-os
[12] http://www.mozilla.org/en-US/firefox/os/
[13] http://www.sencha.com/products/touch/
[14] http://phonegap.com/
[15] http://jquerymobile.com/

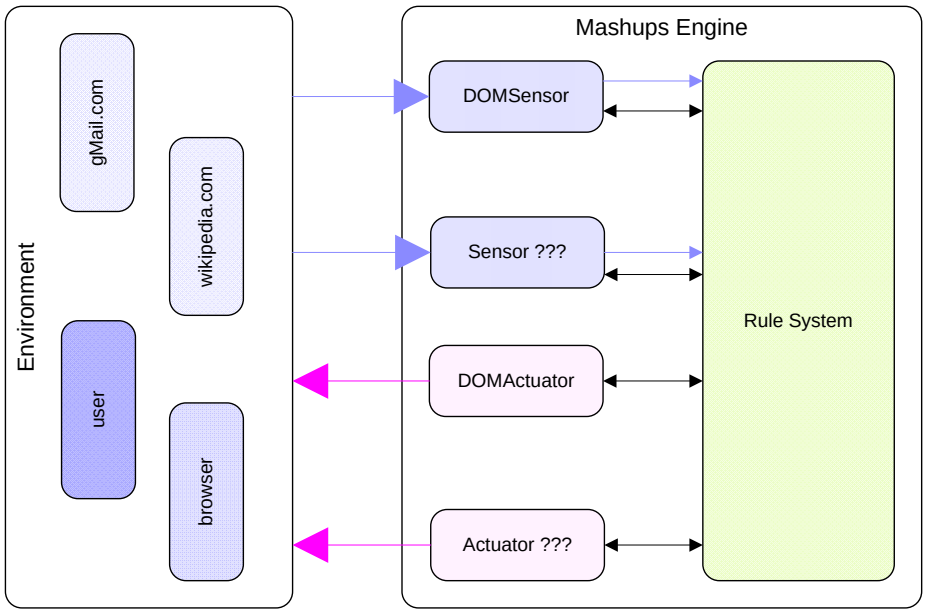


Figure 9: Mashups Engine

les services comme de l'extérieur d'une boîte. De l'intérieur de la boîte on ne voit pas trop. Mais de l'extérieur de la boîte, on peut voir tout ce qui se trouve à l'intérieur de la boîte ainsi que la boîte elle-même. Ici on applique le même principe.

En principe, les pages web accédées par un navigateur ont quelques couches. Il y a les couches JavaScript et Ajax. Le code JavaScript peut être partagé dans quelques fichiers et chargé de plusieurs locations. La même règle s'applique pour les objets Ajax. Ensuite, il y a le couche Modèle Objet du Document (en abréviation DOM en anglais) [Hors *et al.*, 2004]. N'importe ce que la représentation est au sommet, à l'intérieur du navigateur cette représentation est un DOM. C'est la représentation unifiée de l'information que nous avons souligné comme une caractéristique fondamentale pour pouvoir mettre en œuvre un système conforme à notre approche. Dans certains cas (par exemple Firefox[16] utilise XUL [17]) même le navigateur lui-même est un énorme DOM. Un aspect important sur le DOM est qu'il est complètement basé sur l'événement. Toute interaction, toute modification est signalée par un événement DOM [Pixley, 2000]. En conséquence, nous avons aussi la deuxième condition fondamentale, que nous avons identifiée alors qu'on enquêtait sur l'appareil TomTom. Une troisième couche est la Feuille du Style en Cascade (en anglais CSS) [Bos *et al.*, 2010]. Cette couche applique le style. La quatrième couche est la représentation: soit une page web de base, un document XML, ATOM, etc. Le moteur de mashup est une boite comportant toutes ces couches. Ainsi, il a accès à chacun d'entre eux. Toute interaction, conversation, collaboration qui existe entre l'utilisateur, le navigateur et l'un des services en créant un mashup, est réalisé par des actions et des événements. L'ensemble des actions et des événements forment le comportement. L'utilisateur est représenté dans le système par l'intermédiaire de son comportement (actions et événements).

Le moteur de mashup représenté dans la figure 9 comporte un ensemble de capteurs et un ensemble d'actionneurs. Par les capteurs, le moteur perçoit les interactions et les communications avec l'environnement. Le moteur utilise des actionneurs pour modifier et / ou de communiquer de nouveau

[16] *http://www.mozilla.com/en-US/firefox/*
[17] *http://developer.mozilla.org/En/XUL*

avec l'environnement. L'environnement comprend un nombre arbitraire de services et de mashup, le navigateur et l'utilisateur (par ses interactions avec le système).

Chapitre 5 Résumé – Exécution

Ce chapitre concerne l'exécution d'une application Web suivant l'approche et le modèle conceptuel que nous avons introduit dans le Chapitre 3. Pour récapituler, l'exécution est fondée sur des règles et processus et est adaptée pour les navigateurs web.

Le Model Objet du Document ainsi que les Evénements DOM sont des interfaces neutres de plateforme et langage qui permettront aux programmes et scripts d'accéder de la façon dynamique et d'actualiser le contenu, la structure et le style des documents, et respectivement, permettent l'enregistrement des gestionnaires des événements, décrivent la circulation de l'événement par une structure type arbre et fournissent l'information contextuelle de base pour chaque événement.

Par conséquent, notre moteur d'exécution utilise directement ces interfaces neutres de plateformes et langage. Ainsi, notre moteur d'exécution est neutre du point de vue de la plateforme et du langage. En outre, nous pensons que grâce à la combinaison d'un moteur d'exécution basé sur des règles et l'utilisation de ces interfaces neutres des plateformes et du langage peuvent être atteints d'une approche unifiée et générique pour la définition et l'exécution de nouvelles applications définies de l'utilisateur final qui respectent entièrement la liste des exigences que nous avons identifié dans la section 2.6.

La composante de raisonnement d'un Système de Règles de Production standard est le Moteur d'Inférence (même dans notre cas). Le Moteur d'Inférence met en correspondance des faits et des données contre les règles de déduire des conclusions résultant des actions. Bien que les règles de production se composent de deux parties principales : (1) conditions et (2) actions, nous traitons principalement avec des règles de réaction ou des règles ECA qui comportent trois parties : (1) événement, (2) conditions et (3) actions. Les deux types de règles utilisent la logique du premier ordre pour la représentation des connaissances.

Le processus de mettre en correspondance de faits nouveaux ou existants contre les règles est appelé algorithme de corrélation et est effectuée par le Moteur d'Inférence. Notre implémentation est une variante de ReteOO qui aborde les systèmes orientés vers l'objet. Plus précisément, ceux basé sur la fermeture (JavaScript). En outre, il prend en charge par default les Evénements DOM.

Les règles pour notre système de règles sont stockées dans le plan défini par l'utilisateur (mashup). La mémoire de travail où toutes les faits résident de sorte que le Moteur d'Inférence peut les accède représente dans notre cas la structure DOM entière (toutes les services, pages web qui ont été associés avec les contextes dans le plan défini par l'utilisateur) o laquelle le Moteur d'Inférence a accès. Dans notre implémentation tous les faits se trouvent déjà dans la Mémoire de Travail. En outre, la Mémoire de Travail dans notre cas tient aussi les Evénements DOM, tant les événements DOM W3C prédéfinies (par exemple `click`, `dblclick`, `mouseout`, `mouseover` etc) ainsi que les événements DOM personnalisés. Toutes les interactions ont lieu à l'intérieur de la Mémoire de Travail. La mémoire de travail est en vie aussi longtemps que le mashup est activé dans le navigateur (cela signifie que tant que dans le navigateur web il y a un onglet ouvert dans le navigateur qui contient le mashup, la mémoire de travail sera maintenu en vie).

Il y a deux méthodes d'exécution pour le système des règles : le chainage avant et le chainage derrière. Notre implémentation est une implémentation à chainage avant.

Les règles sont écrites en employant la Logique du Premier Ordre (FOL en anglais), ou la logique des prédicats qui étend la logique propositionnelle. Les faits sont des objets (par exemple java beans, objets JavaScript). Ainsi, pour notre système de règles, les faits sont tous objets JavaScript auxquels le

moteur a accès, c'est-à-dire tous objets de la Mémoire de Travail. Cependant, on n'emploie que les domaines des objets dans le procès de raisonnement, ou la structure statique d'un objet : propriétés (domaine, propriété ou attribut ont le même sens) et leurs valeurs. Puisque tous les éléments de notre modèle sont de `Concepts`, et en outre, un concept est une sous-classe de la classe `uml::Class`, puis l'un d'eux peut être faits. Dans les règles de conséquence pourrait être utilisé pour raisonner sur l'un d'eux d'une manière unifiée.

Une règle précise que sur l'événement attiré, si une série particulière des conditions intervienne, spécifié dans le côté gauche (LHS en anglais) alors on fait ainsi, étant mentionnée comme une liste des actions dans le côté droit (RHS en anglais). LHS est un nom commun pour la partie conditionnelle de la règle. Elle est constituée d'un zéro ou plusieurs éléments conditionnels. Les éléments conditionnels concernent les concepts, qui à son tour appartiennent aux contextes.

La condition (LHS) partie d'une règle peut comporter quelques éléments conditionnels. Nous envisageons une série d'éléments conditionnels pour notre langue d'exécution:
(1) un `ConceptConditional`, (2) un `JavaScriptBooleanConditional`,
(3) un `EqualityConditional`.

L'élément conditionnel concernant les Concepts (`ConceptConditional`) est les plus important. Ce conditionnel est construit pour accommoder les objets et les caractéristiques des objets. Il décrit en fait la façon dont l'objet (le concept) devrait ressembler. Un objet (concept) comporte un type et propriétés (domaines). Par conséquent lorsqu'il s'agit d'un objet, dans notre cas, un `concept`, nous devons vérifier le `type`, nous avons besoin de pouvoir tester / contraindre les valeurs de propriétés et nous devons être en mesure de lier la valeur d'une propriété d'objet à une variable, de sorte qu'il puisse être utilisé plus tard dans un autre élément conditionnel d'une règle. En outre l'objet lui-même, pas seulement une propriété de celui-ci, peut être lié à une variable.

Conclusions

World Wide Web (WWW) est devenu le plus grand dépôt d'informations que l'homme ait jamais assemblé et il est en croissance continue. Le nouveau WWW ou l'Internet de l'Avenir est celui d'un Internet des Services et un Internet des Objets.

Naturellement, une série des questions se posent à partir de ce contexte : comment filtrez-vous les objets pour créer plus de valeur que vous obtenez actuellement ? Comment pouvez-vous regrouper les objets d'une manière intelligente et facile au lieu de la faire dans votre tête? Le monde ne peut pas être décrit sans ambiguïté, alors comment pouvez-vous permettre aux utilisateurs de traiter avec le monde à leur manière, en fonction de leur compréhension?

On a largement affirmé que la solution vient du droit des participants (utilisateurs finaux). Malheureusement, bien que beaucoup d'efforts ont été mis dans le développement d'un grand nombre de cadres à s'attaquer à certains des problèmes que ce nouvel environnement a apporté (on peut voir le chapitre 2), la conception et le déploiement d'un tel logiciel capable d'interaction directe et de l'autonomisation de l'utilisateur final est toujours un problème.

Notre objectif dans cette thèse est de combler ce manque d'outils qui sont capables d'une interaction directe et l'autonomisation des utilisateurs finaux, d'une façon unifiée. Pour atteindre cet objectif nous proposons une approche centrée sur l'utilisateur.

Pour son implémentation nous avons développé un modèle conceptuel pour cette approche, nous proposons une architecture qui est conforme à l'approche et nous avons aussi proposé un moteur d'exécution qui emploient règles ECA et les processus pour exécuter les applications que les utilisateurs finaux ont défini comme mushups.

Nous soutenons que par l'approche discutée tout au long de cette thèse que nous avons beaucoup avancé vers une approche entièrement centrée sur l'utilisateur permettant aux utilisateurs finaux de créer leurs propres applications en utilisant Architectures Orientées vers les Services.

Nous envisageons quelques directions possibles pour étendre le travail que nous avons présenté et discuté au sein de cette thèse : (1) modélisation visuelle des plans définis par l'utilisateur ; (2) l'utilisation de l'approche des platesformes mobiles ; (3) ajout de différents types de raisonnement comme la programmation logique abductive.

De notre point de vue, le chaînon manquant nécessaire pour obtenir un système d'utilisateur final complet est une approche visuelle (sur la base de modèles de conception d'interaction) qui cachera complétement tous les autres aspects qui exposent encore des aspects techniques connexes, tels que le mashup, rédigé en un format JSON.

Chapter 1

Introduction

World Wide Web (WWW) has become the greatest repository of information that man has ever assembled and it is continuously growing. WWW transformed itself into a generative environment that fosters innovation through the advance of technologies and a shift in people's perception of the Web and how they use it [Zittrain, 2008]. It "has shifted from transaction-based Web pages to interaction-based ones" [Ogrinz, 2009]. WWW changed radically, also the way knowledge is shared, by lowering the barrier for publishing and accessing documents [Bizer *et al.*, 2010]. Moreover with the proliferation of Web APIs the Web has become a highly programmable platform [Aghaee *et al.*, 2013].

The new WWW or *Future Internet* is that of an Internet of Services and Internet of Things. As described by SAP co-CEO Henning Kagermann [Kagermann, 2008],

> The Internet of Services is largely based on a service-oriented architecture (SOA), which is a flexible, standardized architecture that facilitates the combination of various applications into inter-operable services.

The notion of a service is defined by Lovelock et al. [C. Lovelock, 1996] as

> an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production.

The first steps towards such an approach were made by proposing the notion of a Web service. A Web service is a form of a service. The notion of a Web service we use hereafter is provided by Sommerville [Sommerville, 2006]:

> loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols.

The concept of Service Oriented Architecture (SOA) is probably the most popular outcome of research in service based composition and has created a completely new way of composing existing functionality through dynamically binding and invoking services into a composite application. As stated in [Ogrinz, 2009], SOA is an evolutionary milestone and not a revolutionary one, where the understanding of a *service* in SOA is that of a *business task*. Such tasks are implemented in environments that facilitate loose coupling.

Yet, in reality, because of their still complicated technical nature, "Web services have hardly been adopted beyond the boundaries of enterprises" [Davies *et al.*, 2009].

According to [O'Reilly, 2007], "Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as a platform, and an attempt to understand the rules for success on that new platform". Web 2.0 technologies are interactive and require users to generate new information and content or to edit the work of other participants [Chul *et al.*, 2009]. Chul et al. continue to state in [Chul *et al.*, 2009] that, "the right solution comes from the right participants".

Naturally, a series of questions arise from this context: how do you filter things to create more value than you currently get? how do you aggregate things in an intelligent and easy way instead of doing it in your head? The world cannot be described unambiguously, so how can you allow users to deal with the world in their own way, based on their understanding? Levine in his book "Cluetrain manifesto" [Levine, 2009] was arguing that markets are conversations so how can users be involved in the conversation? how can users be empowered with easy consumption of the services, information, things that they found around?

By Levine [Levine, 2009], whether delivering information, opinions, perspectives, dissenting arguments or humorous asides, humans conversations are done typically in an open, natural, uncontrived manner. Future Internet (Web 2.0), has opened the ways towards enabling conversations among human beings that were simply not possible before.

However design and deployment of such software capable of direct interaction and empowerment of the end-user is still an issue. We have on one side *users* that have ideas, but do not have technical background and lack programming skills to do the development by themselves. On the other side, we have large amounts of *data, resources and services* that could be aggregated both in terms of data, but most important in terms of behavior to innovate and create new things. Nardi underlines this aspect clearly in [Nardi, 1993] stating that

> "we have only scratched the surface of what would be possible if end users could freely program their own applications... As has been shown time and again, no matter how much designers and programmers try to anticipate and provide for what users will need, the effort always falls short because it is impossible to know in advance what may be needed... End users should have the ability to create customizations, extensions and applications..."

1.1 Objective of this thesis

Our goal in this thesis is to address this lack of tools that are capable of direct interaction and empowerment of end-users, in a unified manner. We are concerned specifically with Web based tools. We assert that for such web based user-centric systems, users, services (in the general form as defined in [C. Lovelock, 1996]), semantics and context, are key components of the system.

Context, as defined by Dey and Abowd [Dey et Abowd, 1999a]

> 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 the applications themselves.

Lately, the notion of context has been considered not simply as state but as part of a process in which users are to be involved [Coutaz *et al.*, 2005]. Context greatly influences the way humans or machines act, the way they report themselves to situations and things; furthermore, any change in context causes a transformation in the experience that is going to be lived [Bolchini *et al.*, 2007]. Many psychological studies have shown that when humans act, and especially when humans interact, they consciously and unconsciously attend to context of many types as stated in [Grudin, 2001].

Harnessing collective intelligence [O'Reilly, 2007] by allowing people to reuse data is a concrete example that same data could gain a different meaning and could provide a different perspective on a particular matter based on context or based on users' goals.

See for instance the book store example given by David Weinberger in his book entitled "Everything Is Miscellaneous: The Power of the New Digital Disorder" [Weinberger, 2007]. While the normal setup of a book store might work fine for users that know what they want, the same setup might not work for others who do not really know what they want. Weinberger calls the first category *seekers* and the second one *browsers* and emphasizes the fact that there are almost as many ways to organize for browsers as there are browsers [Weinberger, 2007].

This example nicely underlines the aspects we have already emphasized: i.e. (1) each *browser=user* has its own way to resolve its problem, or has its own idea on how to organize data; (2) from this uniqueness that each user has, we can end up with different ways of combining services and data, in order to achieve new perspectives.

Personalization technology is used to dynamically change / provide / suggest content that is relevant to users. Contextual information, such as *location* (see for instance [Raptis *et al.*, 2005]) has been traditionally used in personalization and recommender systems. However this does not necessary reflect or fulfill users' expectations. Advertising in social networks is one example of personalization. Ads are based on many factors, i.e. network of friends, pages that someone likes, brands that someone likes etc. Personalization is one possible use case for our approach.

1.2 Our approach

Mashups are one of Web 2.0 paradigms. The term *mashup* has been borrowed from music and is a song or composition created by blending two or more songs[1]. In web's case a mashup (see for instance Altinel et al. [Altinel *et al.*, 2007]) has been defined mostly from a technological perspective as a hybrid application that uses and combines data, presentation or functionality from two or more sources to create new services, frequently by means of Web APIs.[2].

Conferences calendar is just an example – out of many – that requires a mixture of the ideas (users, services, data, behavior, context) discussed here to make it possible. We will be using this use case as a recurring example, through out this thesis to explain our approach.

Such a calendar is user specific, since for example some users might be interested in web related conferences, others in semantic web, others in rules or business processes conferences. Contextual information is mashed up to fulfill some user's goal, hence specific information about conferences is stored in a calendar context. At least two services are required: one that deals with conferences and one that offers a calendar. From a technological point of view these services might not be compatible with each other.

For scientists in the IT field, the DbWorld Mailing List is the well known place where they can search for an IT conference. A series of information are provided here, but most important are the subject, deadline and the web page of the event published. From a technical perspective, DBWorld does not provide an API to allow programmatic access and interrogation of the service. In consequence, with respect to current mashups approaches, this service is useless.

On the other hand, Google Calendar is one of the most known Google Apps services. The information of interest for a calendar is the title of the event, the date and description of an event. This information is found in a Google Calendar. Opposed to the DbWorld services, Google provides for this

[1] *http://en.wikipedia.org/wiki/Mashup_(music)*, retrieved 9 December 2013

[2] *http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)*, retrieved 9 December 2013

service, besides the regular web page representation, an API to access the contents. This situation is a just a simple example that sustains the need to be able to deal with non-uniform ways of service access.

The usual way to achieve this goal of having conferences stored in Google Calendar by their deadline requires manual interactions: (1) the user is required to maintain two open tabs in the browser; (2) even though there might be several entries that comply with a search term, the user must deal with the events one by one as DBWorld does not provide built in search functionality; (3) the user has to move between the open tabs several times, in order to store only one event in the calendar, since just one piece of information can be copied and pasted at a time (e.g. the title of the event).

Recorder/play like tools cannot be used in this scenario because the application must react according to user's behavior and according to specific search results. For example a particular search might return 5 entries while another might return 2 entries. Current mashups approaches are also unable to address these type of use cases for several reasons, i.e.

1. current approaches are most of them data-based (data means only RSS/ATOM);
2. current approaches are mostly API based (REST services) or WSDL based services;
3. current approaches that try to take into account behavior are restricted to predefined actions;
4. current approaches are platform specific, meaning that in most of the cases, users are required to run heavy infrastructure; users are required to install servers, configure them, maintain them and so forth.

1.3 The GPS device metaphor

While currently there are no generic systems and approaches that, to the best of our knowledge, could allow the design and deployment of use case such as the conferences calendar, there are systems that provide similar behavior with the one we are looking for in other fields. Figure 1.1 depicts a system used almost daily. The goal of this system is to help and guide an user to get from place A to place B, for example to get from Paris to Berlin.

This system incorporates a driver, a GPS device, a car, satellites, and maps. While the context is clear and concerns navigation, the *goal* differs from one user to another as each user has its own starting point and its own destination. To point the way, the GPS devices requires maps. Maps are understood both by the user as well as the GPS device and provide context-dependent information (e.g. the information and the amount of information provided inside a town is different than the one outside a town). There is a continuous interaction between the user and the GPS device. But to actually fulfill the goal of getting to destination, the driver must actually drive the car, there is no other way. Hence the driver itself is part of the system.

Interestingly enough, this system can deal both with *seekers* as well as *browsers*. For seekers the GPS device can point the way towards destination, for browsers can give the current location and as such can guide them towards finding what they are looking for. Think about the case when you are in the city (Paris) somewhere in Montmartre and you are looking for a nice restaurant. Using a GPS device with Google Maps you can find this type of information. It can deal with on demand goals according to drivers' needs and provides information in strong connection with the context.

We consider that a system with similar characteristics it is also required to address user-centric systems that integrate data, services and behavior available in the Web 2.0 [Pascalau, 2011b]. This type of software system has *people not processes* [Collins, 2008] at its heart and is capable of tackling

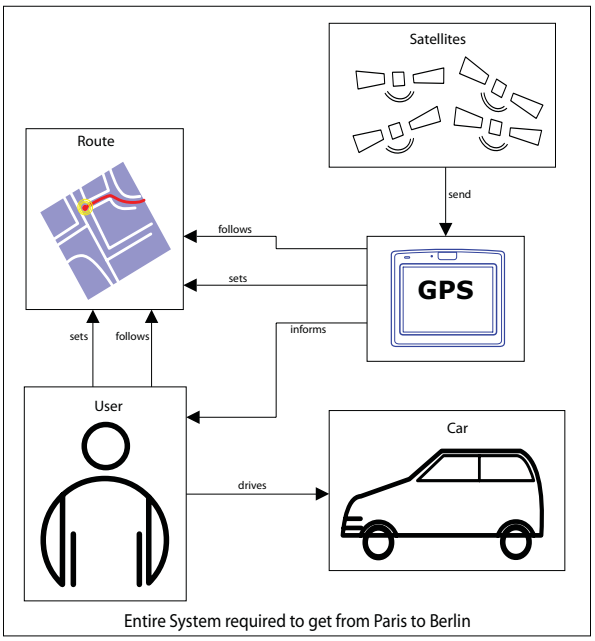


Figure 1.1: GPS Example

software on demand, business intelligence on demand, instant use but powerful, innovation and creativity.

1.4 Contributions

The contents of this thesis reside at the confluence between the following topics: Software Engineering (SE), Service Oriented Architectures (SOA), Context-awareness, Semantic Web and Reasoning, Business Process Management (BPM) and Information Systems.

The work presented throughout this thesis is presented primarily from a SE perspective. There is little focus on building theories and making them explicit [Sjøberg *et al.*, 2008, Easterbrook *et al.*, 2008] in SE and because we employ knowledge from several topics, as we have previously stated, we find important to state that the philosophical stance [Easterbrook *et al.*, 2008] we adopt for our research method is *pragmatism*.

> Pragmatism acknowledges that all knowledge is approximate and incomplete, and its value depends on the methods by which it was obtained [Easterbrook *et al.*, 2008, Menand, 1997].

For pragmatists, truth is what works at the time, entailing a degree of relativism. This means that what is useful for one person might not be useful for another; therefore truth is relative to the observer [Easterbrook *et al.*, 2008]. Pragmatism adopts an engineering approach to research that values practical knowledge over abstract knowledge. Mixed research methods are used to shed light on the issue under study.

Our main contribution is the development of a holistic approach for web based systems that are user-centric and that integrate data, services and behavior available on the Web 2.0.

We believe that the approach we have developed gets us one step closer towards completely allowing end users to program their own applications. Moreover the approach we have developed comes to complement the existing techniques.

Currently the software development processes in Software Engineering are mainly focused on modeling and implementing software from a system-centric perspective – hence the design and implementation process are centered around the structure of the software system being built and interactions between its components. We believe that in Future Internet' applications, the focus needs to be on the end-users.

This trend towards empowering end-users and bringing them into the development process is becoming more and more visible, i.e. [Harris, 2013, Aghaee *et al.*, 2013, Aghaee et Pautasso, 2013]. For instance Derrick Harris talks in [Harris, 2013] about a new Microsoft 'design-first' development strategy for designing software that business users actually want to use.

In details, our contributions are presented in several steps:

- **Step 1:** We start by addressing the conceptual aspect of the problem. We revisit current web technologies with an accent on mashups applications from a conceptual point of view and discuss the concepts we are going to use in our approach.
- **Step 2:** We propose an architecture that inherits characteristics from the TomTom device metaphor, and considers the user as being a component of the system. Principles of SOA, SaaS and Web are mixed together in a hybrid solution.
- **Step 3:** We develop the operational semantics of the system.
- **Step 4:** We discuss the implementation of such systems, provide a set of use cases, and describe a running prototype which constitute a proof of concept of our approach.

Each step is presented in a dedicated chapter.

Chapter 2 is devoted to discussing related work. Different mashups approaches are discussed together with the influential trends and technologies that formed the groundwork for mashups development. We discuss also the reason for which we consider mashups as pertinent related work and we expose our opinion about why some of the mashups techniques have been discontinued or have been integrated into larger projects. We conclude the chapter with a list of requirements with which the approach and system that we are going to introduce in the next chapters need to comply with.

Chapter 3. In this chapter we present our conceptual approach. This approach is strongly end-user oriented. The approach foresees a composite system that comprises a human user and an intelligent system. We discuss the conceptual aspects that drive our approach: end-user / user-centric; plan; two layer system; intelligent system; human - system interaction. Second part of the chapter is describes the conceptual model associated with our approach.

Chapter 4. We propose an architecture for the systems that we discuss throughout this thesis. This architecture follows the TomTom metaphor. It is influenced by the Newell's [Newell, 1994] Model Human Processor and inherits from Service Oriented Architecture (SOA) and Distributed Oriented Architecture (DOA).

Chapter 5. We propose an execution model based on a forward chaining reasoning technique.

Chapter 6. This chapter asserts a series of implementation guidelines on how such systems should be implemented. A series of use cases are presented in this chapter. These use cases will be used to validate our approach. We discuss also to which extent the requirements we identified in Chapter 3 have been fulfilled. A running prototype of our approach will be described in this chapter as well.

Chapter 7. Conclusions of the work presented in this thesis are enumerated in this chapter together with future steps on how this work can be improved.

Here is a list of the most representative publications that sustain this thesis:

1. Emilian Pascalau (2011). Mashups: Behavior in Context(s). Proceedings of the 7th International Workshop Knowledge Engineering and Software Engineering (KESE7) collocated with CAEPIA 2011 (Conference of the Spanish Association for Artificial Intelligence), Tenerife, Spain. CEUR Workshop Proceedings, vol. 805 (double blind peer review)

2. Emilian Pascalau (2011). Towards TomTom like Systems for the Web - A Novel Architecture for Browser-based Mashups. BEWEB 2011 March 25, 2011, Uppsala, Sweden, ACM.

3. Emilian Pascalau and Clemens Rath (2010). Managing Business Process Variants at eBay. In the 2nd International Workshop on BPMN, Potsdam, Germany.

4. Emilian Pascalau and Adrian Giurca (2009). A Rule-Based Approach of Creating and Executing Mashups. In Proceedings of 9th IFIP Conference on e-Business, e-Services, and e-Society, (I3E 2009). C. Godart et al. (Eds.): I3E 2009, IFIP AICT 305, pp. 82-95, 2009. 23-25 September, Nancy, France.

5. Emilian Pascalau and Adrian Giurca (2009). A Lightweight Architecture of an ECA Rule Engine for Web Browsers. In Proceedings of 5th Knowledge Engineering and Software Engineering, KESE 2009, collocated with KI 2009, G. J. Nalepa and J. Baumeister(Eds.), September 15, 2009, Paderborn, Germany, CEUR Workshop Proceedings, vol. 486.

6. Adrian Giurca and Emilian Pascalau. JSON Rules. In Proceedings of the 4th Knowledge Engineering and Software Engineering, KESE 2008, collocated with KI 2008, G. J. Nalepa and J. Baumeister(Eds.), Spetember 23, 2008, Kaiserlautern, Germany, CEUR Workshop Proceedings, vol. 425.

Chapter 2

Related Work on Mashups

In this chapter we review existing work in the area of Web services, Web 2.0, Semantic Web, Semantic Web Services, Web of data presenting their influences on developing mashup tools.

To the best of our knowledge we are not aware of any other survey of such dimensions and which addresses so many aspects related to mashups development.

Mashups are one of the Web 2.0 paradigms and a promising End User Development (EUD) application area [Grammel et Storey, 2008]. We believe that they come most closely to the type of use case that we have enunciated in Chapter 1.

The term *mashup* is borrowed from pop music, where it denotes remixing songs or parts of songs to deliver new derivative works. Similar to this, Web mashups remix content from the Web and deliver new results, often insights among disparate and originally independent sets of information and function, and thus create innovation through assembly. Mashup developers identify particular opportunities from combining existing capabilities on the Web and offer new solutions and inspiration that in turn lead to new and innovative opportunities.

Several definitions have been given to define the concept of *mashups*. Altinel et al. [Altinel *et al.*, 2007] define a mashup as

> a web application that combines content from two or more applications to create a new application. Situational applications are enterprise web applications built on-the-fly to solve a specific business problem. They are often developed without involvement of the IT department and operate outside of its control. They combine data from a variety of enterprise sources such as SAP or Office applications, back-end databases, and content management systems.

In a similar manner Grammel and Storey [Grammel et Storey, 2008] define mashups as

> applications that reuse and combine data and services available on the web. They are developed in a rapid, ad-hoc manner to automate processes and remix information. This enables users to explore information in new ways and saves valuable time that may be lost in laborious routine tasks.

Although all the definitions provided emphasize more or less the same characteristics, Eric Schmidt from Google defines such novel applications as

> "applications that are pieced together" - with the characteristics that the apps are relatively small, the data is in the cloud, the apps can run on any device (PC or mobile), the apps are

very fast and very customizable, and are distributed virally (social networks, email, etc).[1]

2.1 Mashups and Web Services / SOA

The idea of providing services on the web has been tightly bound to Web services technologies [Pedrinaci et Domingue, 2010]. The reality, however is that, Web services, despite their name, are hardly a Web-oriented technology, but rather one that is enterprise oriented. Moreover because of their still complicated technical nature, "Web services have hardly been adopted beyond the boundaries of enterprises" [Davies *et al.*, 2009].

The notion of a service is defined by Lovelock et al. [C. Lovelock, 1996] as

> an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production.

A Web service on the other hand has been defined in [Sommerville, 2006]:

> as a loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols.

The concept of Service Oriented Architecture (SOA) is probably the most popular outcome of research in service based composition and has created a completely new way to composing existing functionality through dynamically binding and invoking services into a composite application. As stated in [Ogrinz, 2009], SOA is an evolutionary milestone and not a revolutionary one, where the understanding of a *service* in SOA is that of a *business task*. Such tasks are implemented in environments that facilitate loose coupling.

A holistic set of standards [MacKenzie *et al.*, 2006] has been developed to support the automatic discovery of service, binding of a service provider and a service requestor and invoking service capabilities through homogeneous interfaces. These are the Web Service [Booth *et al.*, 2004] standards along with specification of WSDL [Christensen *et al.*, 2001] and SOAP [Gudgin *et al.*, 2007].

The Universal Business Registry part of the Universal Description Discovery and Integration (UDDI) [Hately *et al.*, 2004] is probably the most well known effort towards supporting publication and later discovery of services on the Web. Unfortunately, same as the Web services themselves these registries did not have too much success either. They were complex registries and they lacked support for expressive queries [Pedrinaci et Domingue, 2010].

To simplify the standard Web Services stack and impelled by the Web 2.0 phenomena Web APIs have emerged, also called RESTful services when they comply with the REST [Fielding, 2000] architectural style. These services rely on a different stack that is well suited for the Web: URIs, HTTP, XML and JSON.

Figure 2.1 depicts the general operation required by SOA. On the other hand Figures 2.2 and 2.3 present the mashup development scenarios [Cappiello *et al.*, 2011]. We can easily observe the similarities between the two approaches. Based on the three figures (see 2.1, 2.2, 2.3) both cases (Web services and Mashups) follow the same conceptual approach. In the Web services' case we have an UDDI registry where service providers publish Web services. Service requestors then can find services in the registry and bind to a found service. Similarly in the mashups case services (APIs) are published

[1] http://www.youtube.com/watch?v=T0QJmmdw3b0, retrieved 17 December 2013

in a repository. The repository in this case, does not follow any strict rules or guidelines like in the UDDI case. The WWW could be the repository. Later either developers themselves or users find and mashup (bind) services. However SOA generally focuses on server-side architecture and internal corporate resources [Ogrinz, 2009].

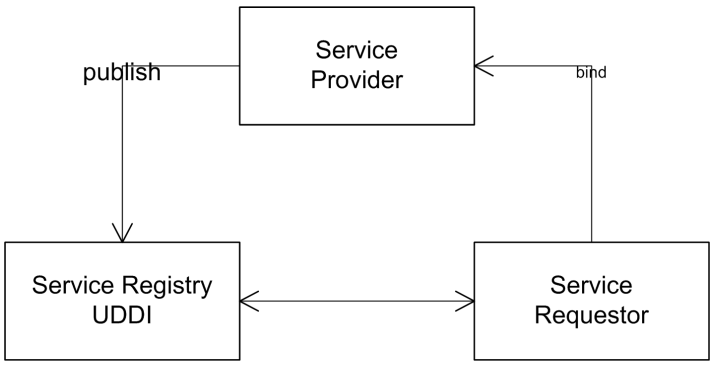


Figure 2.1: SOA operations: Publish, Find, Bind

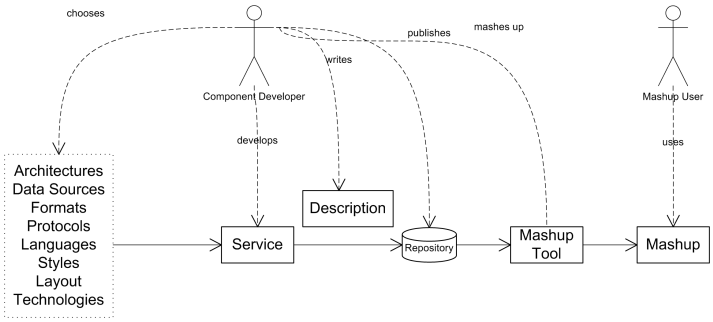


Figure 2.2: Mashup development scenario 1, [Cappiello *et al.*, 2011]

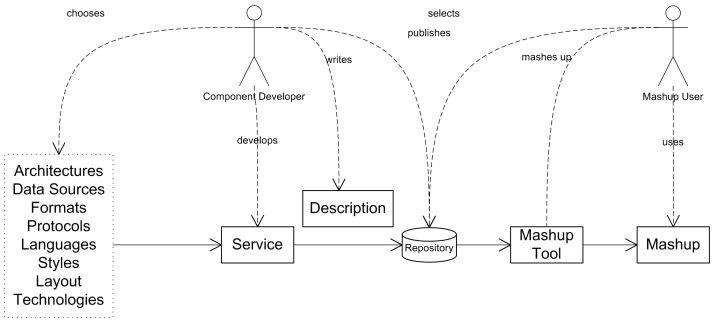


Figure 2.3: Mashup development scenario 2, [Cappiello *et al.*, 2011]

2.2 Semantic Web and Web of Data

The Semantic Web or the web of Linked Data can be an extension of the human-readable Web by adding formal knowledge representation such that intelligent software could reason with the information in an automatic and flexible way.

The web of data is based on some simple principles [Pedrinaci et Domingue, 2010]: (1) every piece of data should be given an HTTP URI, when looked up (2) should provide useful information using standards like RDF and SPARQL; (3) data should be linked to other resources therefore allowing humans and computer to discover new information.

The most notable example of Linked Data and Semantic Web is DBpedia [Auer et Lehmann, 2007]. DBpedia is basically a copy of Wikipedia that takes advantage of "structured data" in the infobox that is found on most of the wikipedia pages. This information is used to automatically extract, annotate and store it as RDF triples.

2.3 Mashups and Software as a Service

While service composition and service based applications address provisioning of a business task from a technological perspective, Software as a Service (SaaS) takes this approach one step further and offers complete applications over the Web [Foster et Tuecke, 2005]. In fact, SaaS can be seen as a business model. Under such a business model, companies would not be required to invest money in an internal development and management of applications. Instead they would rent needed functionality from external service vendors. Usually, end users interact with these applications via a Web browser.

There are some key points that characterize *software as a service* (see for example [Traudt et Konary, 2005]):

- network-based access and management of commercially available software;
- activities managed from central locations rather than at each customer's site;
- enabling customers to access applications remotely via the Web, without the need to install any software on client machines;
- application delivery typically closer to a *one-to-many model* (single instance, multi-tenant architecture) rather than to a *one-to-one model*, including architecture, pricing, partnering, and management characteristics;
- continuous improvement and updates of the hosted software, which obviates the need for end-users to download patches and upgrades;
- frequent integration into a larger network of communicating software—either as part of a mashup or as a plugin to a platform as a service.

We provide a comparison [Pascalau et Giurca, 2009b] that illustrates similarities between the conceptual architecture of desktop applications, Software as a Service, and Mashups.

Computer systems run an operating system, and, on top of this, a number of application programming interfaces (APIs) to access different services of the operating system, e.g., network, display, file system, as depicted in Figure 2.4. These APIs facilitate application development, because they abstract from the mechanisms to interact with hardware, which makes software development easier and more efficient.

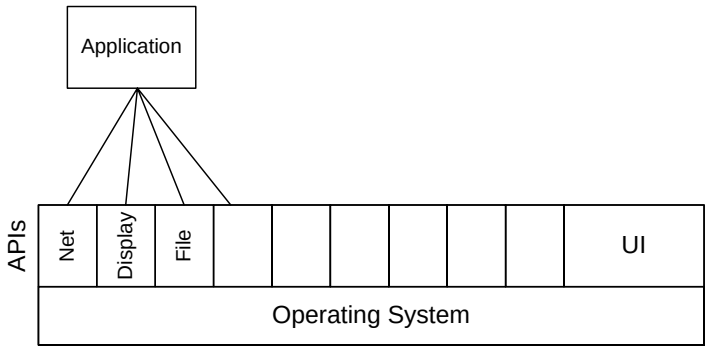


Figure 2.4: Desktop Computer, [Pascalau et Giurca, 2009b]

Software as a Service follows a similar approach, cf., Figure 2.5, whereas computer and operating system have been replaced by Web application servers and the operating system APIs by Web APIs—services that offer access to certain functionality over the Web. Application logic as well as data remain on the Web; applications are not anymore desktop based but run in the Web browsers of users. In this scenario, the APIs are under strong control of the provider of the SaaS application. This is due to the nature of this business model: The provider needs to provide reliable and trustworthy applications. Services and application are provided and controlled by the same authority, designed for a specific and narrow use case. Famous examples of SaaS are many Google products such as Gmail, Docs and Spreadsheets, or Calendar.

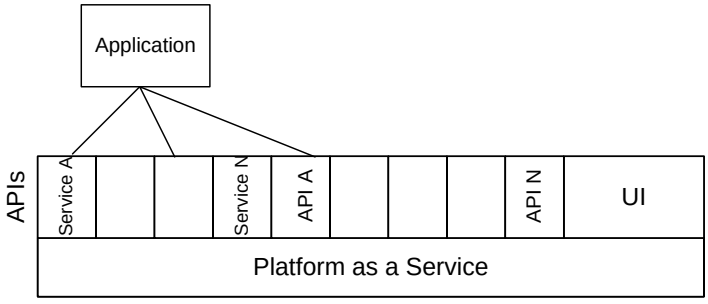


Figure 2.5: Software as a Service, [Pascalau et Giurca, 2009b]

Mashups take the approach of offering Software as a Service another step further and broaden the scope and origin of services [Ogrinz, 2009]. Instead of assuming a closed domain of responsibility to access services, or APIs, on the Web, the Internet has become the operating system—a famous observation that has emerged along with Web 2.0 [O'Reilly, 2005]. While SaaS is mainly based on a proprietary central platform, which offers different services, mashups are generally based on various service sources available on the Web, depicted in Figure 2.6. Many of such services are offered by companies, such as Yahoo, Google, Technorati, Amazon, etc. In other cases, mashups may exploit information or functionality that was not even intended for reuse in other applications[2] [Alba *et al.*, 2008].

[2]In fact, the developer of http://housingmaps.com, Paul Rademacher, was hired by Google to develop the Google Maps API after he leveraged Google Maps by hacking them to create his very famous mashup.

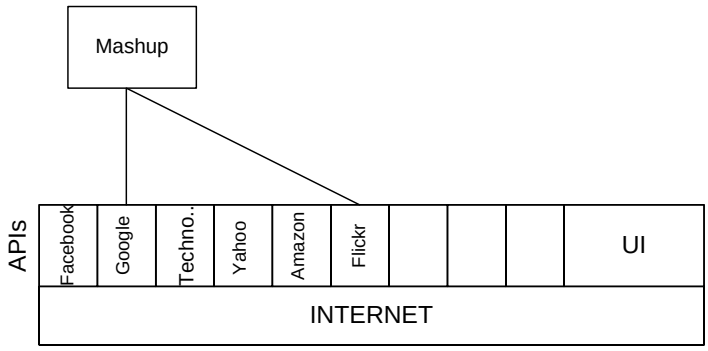


Figure 2.6: Mashup, [Pascalau et Giurca, 2009b]

Similar to the above models, mashups leverage APIs to access data and functionality on the Web; they even use similar technologies and are based on the same paradigms to building applications and interacting with services running on a Web server [Garrett, 2005]. However, services are used regardless of their origin or their intended use case. Mashup developers and service providers are, in general, different bodies, and neither can assume the particular use cases of the other, which led to the development of application independent APIs and protocols. This way, new applications are created from old ones.

Due to their character, mashups expose characteristics beyond those stated for SaaS above:

- Mashups use APIs from different platforms to aggregate and reuse content;
- Usually mashups operate on content and protocols based on open standards, such as Atom [Nottingham et Sayre, 2005], AtomPub [Gregorio et de hOra, 2007], RSS 2.0 [RSS, 2009], and RDF [Klyne et Caroll, 2004];
- "melting pot" style of content recombination, such that, content is aggregated arbitrarily serving situational needs;
- data access is preferably based on the principles of REST [Fielding, 2000] applied to HTTP.

2.4 Mashups and Portals

Data aggregation from multiple sources inside and outside the boundaries of the work place is not a new concept. This approach has been driven by companies' needs of providing users – customers as well as employees – with the ability to access disparate applications or at least services from one single, centralized point: Web Portals or Enterprise Portals. Portals are an established technology that are available as an extension to traditional Web applications [Abdelnur et Hepper, 2003]. In general, a portal aggregates fragments of data in so-called portlets.

A portlet

> is an application that provides a specific piece of content (information or service) to be included as part of a portal page. It is managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to information systems [Hepper, 2008].

A portlet generates a markup fragment, e.g., XML, HTML, that complies with specific rules. Such fragments are aggregated to form a complete document, in the form of a Web page—the application portal, which typically provides following features [Ogrinz, 2009, Sun Microsystems, 2007]:

- **Identity Management:** for example single sign-on (SSO);
- **Secure Remote Access:** centralized administration of what information a user can access;
- **Modularization:** portlets expose information and functionality from disparate systems and may interact with each other;
- **Content Personalization:** users have limited capabilities to modify the layout and presentation of the site;
- **Aggregation and Integration:** the set of portlets offers an aggregate view on a set of information sources and often provides search access among these.

Figure 2.7 depicts an abstract view on portals. A portal consists of a set of portlets, their composition is up to customization on the user's preferences. Portlets are deployed within the same domain boundaries as the portal server, since portlets are rendered on server-side into a composite markup page, e.g., HTML, that is delivered to the user agent, i.e., the browser. The OASIS specification [Thompson, 2008] for remote portlets, describes a way in which portlets could access Web services that are outside of the portal application.

Widget-based mashups — also referred to as "Dashboards" [Kunze, 2009] — represent a specific type of mashups that is quite similar to portals, in that the dashboard offers a set of widgets as information fragments similar to portals offering portlets. The term dashboard refers to dashboards of machines, e.g., vehicles, that provide an overview of a set of gauges at the same time. Yet, each of these gauges retrieves its information from a different origin as the others and is mostly independent of them. These gauges are called widgets or gadgets in the context of dashboard mashups. This paradigm, which will be further addressed in Section 2.5.1, is less restrictive than the portal approach and allows non-programmers to assemble their own mashups or mini applications in a simple fashion. By that, dashboards come close to the ultimate desire of many mashup advocates: making application development end user capable [Janner *et al.*, 2009].

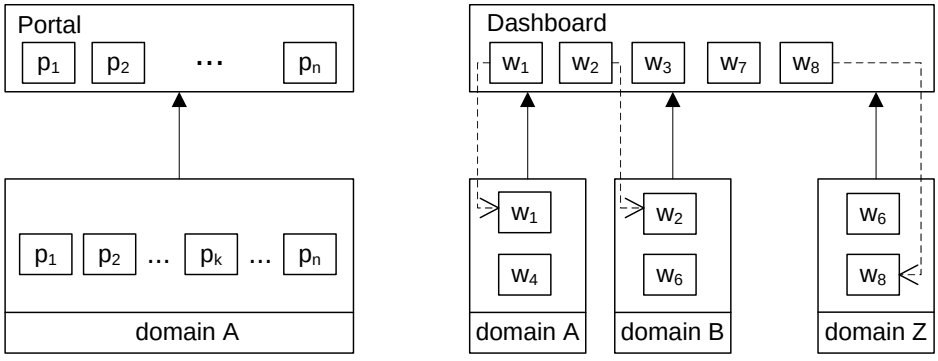


Figure 2.7: Architectural Concepts of Portals (left) and Dashboards (right)

In contrast to portals, where the set of portlets is limited by the driving entity, mainly the organization hosting the portal, dashboards allow the end user to add any widget that they would like—not

only those provided by a dashboard provider. Additionally, widgets can be obtained from any place on the Web that is accessible from the user's computer and are not restricted to those deployed within the same domain[3] as the dashboard itself.

This is illustrated in Figure 2.7: While portals deliver all information fragments (portlets) statically within one single markup document (HTML) from one single domain, a dashboard is only the container for widgets. The widgets themselves are loaded from disparate domains after the instantiation of the dashboard container (indicated through dashed lines in Figure 2.7), and each of them runs within its own execution context independently from each other. Extensions to the container, such as a message based communication hub, cf., [Keukelaere *et al.*, 2008], allow for interaction among widgets. In contrast to portals, dashboards allow end users to add, remove, or change the set of widgets from any source on the Web at runtime, without reloading the container. Widgets need not necessarily comply with regulations, since they are in principle Web applications offered through a URL. This allows widgets to be mashups, in turn.

While mashups tend to get rid of some of the obstacles of portals, e.g., the limited possibilities for customization, and add new flexibility through enabling access to open Web APIs, they suffer particularly from governance issues—identity management and secure remote access—due to the nature of accessing federated content [Kunze, 2009]. Table 2.1 compares some more aspects of portals and dashboards.

Increased interest in dashboards from organizations fueled the development of enterprise level dashboards, often provided through established software providers, e.g., *IBM Mashup Center*[4] or *SAP Rooftop*[5]. At the same time, the formal specification of widgets, their behavior and interaction within a common context is subject to current research, cf., [Abiteboul *et al.*, 2008, Guo *et al.*, 2008, Hoyer *et al.*, 2009, Jackson et Wang, 2007, Keukelaere *et al.*, 2008, Laga *et al.*, 2009, Yu *et al.*, 2007].

Criteria	**Portal**	**Dashboard**
aggregation style	side-by-side	any manner, hybrid content
information fragments aggregated on	server-side	client-side
standards	JSR 168, JSR 268	no standards, but adopted proposals, e.g., Google Gadgets API, W3C Widget Packaging and Configuration
who decides on available content	organization, portal provider	end user

Table 2.1: Portals vs. Mashups—Comparison

[3]The term "domain" is used twofold, here: the domain of responsibility for a set of IT services, which is embraced by an organization; and a domain namespace in terms of the domain name system (DNS) used for referring to resources on the Web [Mockapetris, 1987]. On the Web, both generally complement each other.

[4]http://www-01.ibm.com/software/info/mashup-center/

[5]http://www.sapweb20.com/blog/2010/01/sap-innovation- enterprise-mashup-prototype-rooftop-marketplace/

2.5 Mashups - Conceptual approaches

This section describes the major approaches that have been used to design mashups tools: (1) mashups as collections of widgets; (2) pipes based mashups, and semantic pipes based mashups; (3) hybrid approaches; (4) doman specific languages for mashups.

2.5.1 Mashups as a Collection of Widgets

While the World Wide Web Consortium (W3C) strives to establish a standard definition for widgets—"client-side applications that are authored using Web standards, but whose content can also be embedded into Web documents" [Caceres, 2009]—there are further definitions for the widget concept: Wikipedia defines widgets as

> a portable chunk of code that can be installed and executed within any separate HTML-based web page by an end user without requiring additional compilation. They are akin to plugins or extensions in desktop applications. Other terms used to describe a Web Widget include Gadget, Badge, Module, Capsule, Snippet, Mini, and Flake [free encyclopedia, 2009].

Wicks et al. further characterize a widget as

> a small program or piece of dynamic content that can be easily placed into a Web site. "Mashable" widgets pass events, so that they can be wired together to create something new, [Wicks *et al.*, 2009].

Although the W3C provides a working draft [Caceres, 2009] that deals with packing and configuration of widgets there is no standard for creation and execution of widgets, as discussed in below sections. Companies, such as Google, Yahoo, Amazon, Microsoft, and IBM, have developed their own widgets and widget platforms.

Typically widgets are defined declaratively using an XML based language, but the generated executable code is platform dependent. Run in a Web browser, they are usually embedded using `iframe` elements.

The W3C proposed an architecture for Widgets, cf., Figure 2.8. The concepts involved are:

- **host runtime environment:** the hosting system for mashup instances;
- **media type:** a pre-registered media type [Freed et Borenstein, 1996] that associates the package with a certain runtime environment, e.g., `application/vnd.yahoo.widget`;
- **packaging format:** self-contained, structured resource that encapsulates the contents of a widget;
- **manifest:** a specific resource of the package that contains the configuration of the widget and of the environment;
- **APIs:** a set of programming interfaces that provide functionality specific to widgets;
- **resources:** images, text, graphical user interface markup, style sheets, executable scripts and other possible parts of the widget.

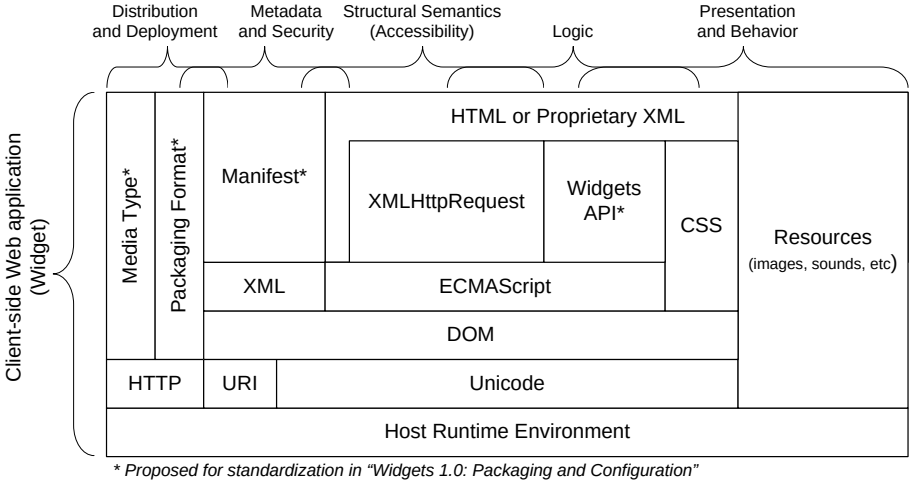


Figure 2.8: Widgets Architecture - W3C, [Caceres, 2009]

Vendor	**Container Format**	**File Extension**	**Media Type**
Yahoo! Widgets Engine	Zip, directory, proprietary flat-file	.widget	application/ vnd.yahoo.widget
Microsoft Windows Sidebar	Zip, Cab, directory	.gadget	application/x-windows-gadget
Google Desktop	Zip	.gg	app/gg
Opera Browser	Zip	.zip	application/x-opera-widgets
Apple Mac OS X Dashboard	Zip, Apple bundle	.wdgt or .zip	application/x-macbinary
AOL Modules	Zip	.html	text/html

Table 2.2: Widgets packaging formats

It is worth mentioning that widgets based on host runtime environment can also be used as desktop applications. However, in this case the host environment is a desktop application that incorporates a mashup server. Such a server, e.g., Konfabulator for Yahoo Widgets[6] has access to the web and also to the local content, binding both contexts.

Tables 2.2 and 2.3 provide information regarding the packaging formats, manifest files for several of the major widget vendors. The mentioned tables are another evidence that there is no accepted standard for widgets.

The next subsections present Google Gadgets and Yahoo Widgets, who, opposed to other approaches, expose an information-centric model. Although there are several vendors tackling widget

[6]http://widgets.yahoo.com/tools/

Vendor	Manifest Format	Manifest File	UI Markup
Yahoo! Widgets Engine	Proprietary XML Yahoo! Widgets Reference	*.kon	Proprietary XMLWidgets Reference
Microsoft Windows Sidebar	Proprietary XML Microsoft Gadgets	gadget.xml	HTML + CSS
Google Desktop	Proprietary XML Google Gadgets	gadget.gmanifest	Proprietary XML Google Gadgets
Opera Widgets	Proprietary XML Opera Config	config.xml	HTML + CSS
Apple Mac OS X Dashboard	Proprietary XML Apple pList	Info.plist	HTML + CSS
AOL Modules (any capable User Agent)	Microformat AOL ModuleT	index.html	HTML + CSS

Table 2.3: Widgets manifest files formats

based technologies, such as Apple with the Mac OS X Dashboard [7], Facebook[8], or Pageflakes[9], their approaches are rather programming-centric.

Google Gadgets.

Google Gadgets [Google, 2009] is Google's approach to tackle mashups. Figure 2.9 and Figure 2.10 depict the information model behind Google Gadgets. Google uses and contributes to the Open Social gadgets specification [OpenSocial, 2009].

Google gadgets rely on two major concepts: `Content` and `ModulePrefs`. A gadget might have more than one `Content` entity. Inside it the user might insert any HTML, possibly with embedded JavaScript, Flash, ActiveX, or other browser objects, specified by the `type` property. A second option is to provide just a URL of a remote page that contains a gadget's content.

`ModulePrefs` specifies the gadget's characteristics. Google gadgets facilitate the use of REST services (Section 2.3), authentication, e.g., `OAuth` entity, and also inherit the views approach (`views` attribute of `Preload`) from portlets [Hepper, 2008].

Google provides means that allow users to create their own dashboard (see Sections 2.4 and 2.5.1): Every Google user has their own iGoogle[10] page where gadgets can be added or removed.

[7] http://developer.apple.com/macosx/dashboard.html
[8] http://developers.facebook.com/
[9] http://developers.pageflakes.com/
[10] http://www.google.com/ig

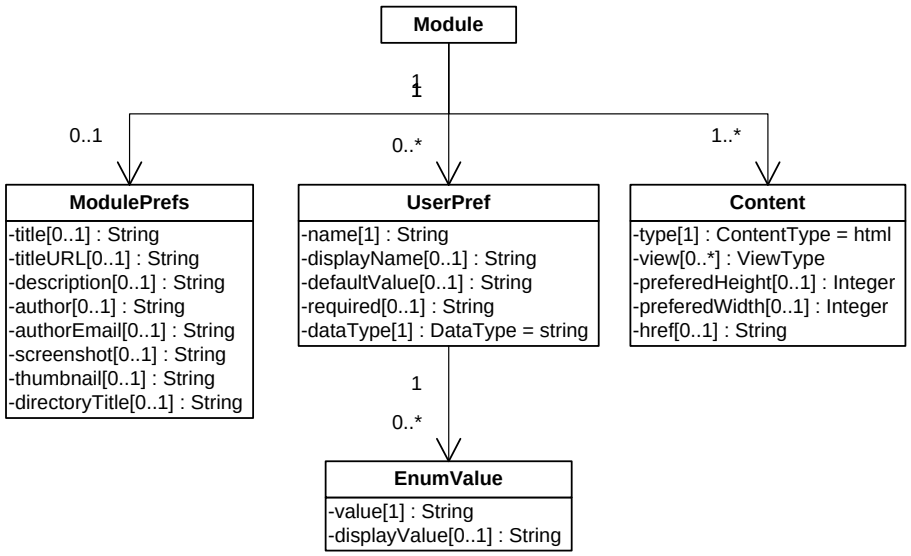


Figure 2.9: Google Gadgets - Main Concepts

Gadgets are stored as XML [Bray *et al.*, 2008] files, e.g., http://www.google.com/ig/modules/calendar3.xml. The XML definitions have to be published on the Google Gadget platform and, only after that, can be processed by the Google Gadget Engine.

From an engineering point of view gadgets are embedded with the help of `script` elements. The result of their execution is usually an `iframe` element that is added to the DOM [Hors *et al.*, 2004] of the embedding page.

Yahoo Widgets.

Similarly to Google gadgets, Yahoo Widgets [Yahoo, 2009]—previously known as "Konfabulator"[11]—have a formal grammar, but a proprietary one that is not striving to become part of an open Web standard. Figures 2.11 and 2.12 depict the Yahoo Widgets concepts. Some of the concepts are inherited from HTML, e.g., `Image`, `Text`, `Frame`, `Textarea`, `ScrollBar`, `Window`, `Canvas`. Concepts for menus, `MenuItem`, and for accessibility, `HotKey`, are available, as well.

There are two fundamental differences compared to Google Gadgets: First, Yahoo Widgets are oriented towards a functional API based approach, providing a `Script` element. Second, there is an obvious tendency for a high level approach concerning actions. `Action` and `Timer` concepts emphasize the issue of actions triggered by events. However, in the Yahoo widgets context they are just part of a programmatic interfaces. On the other hand, from a high level perspective this approach is either related to Event-Condition-Action (ECA) rules, where actions are triggered by events, or to business processes where actions are preceded by concrete events or by implicit events.

The technical perspective has similarities as well as differences to Google. Users can run widgets from their local machine, i.e., using the local server application. In this case an engine is required to be installed on the client machine. Through the installed engine, widgets can access both the local content and the remote content. Widgets can also be published on a Web server, the Yahoo Widget Server, and

[11] http://konfabulator.com/cartoon/partOne.html

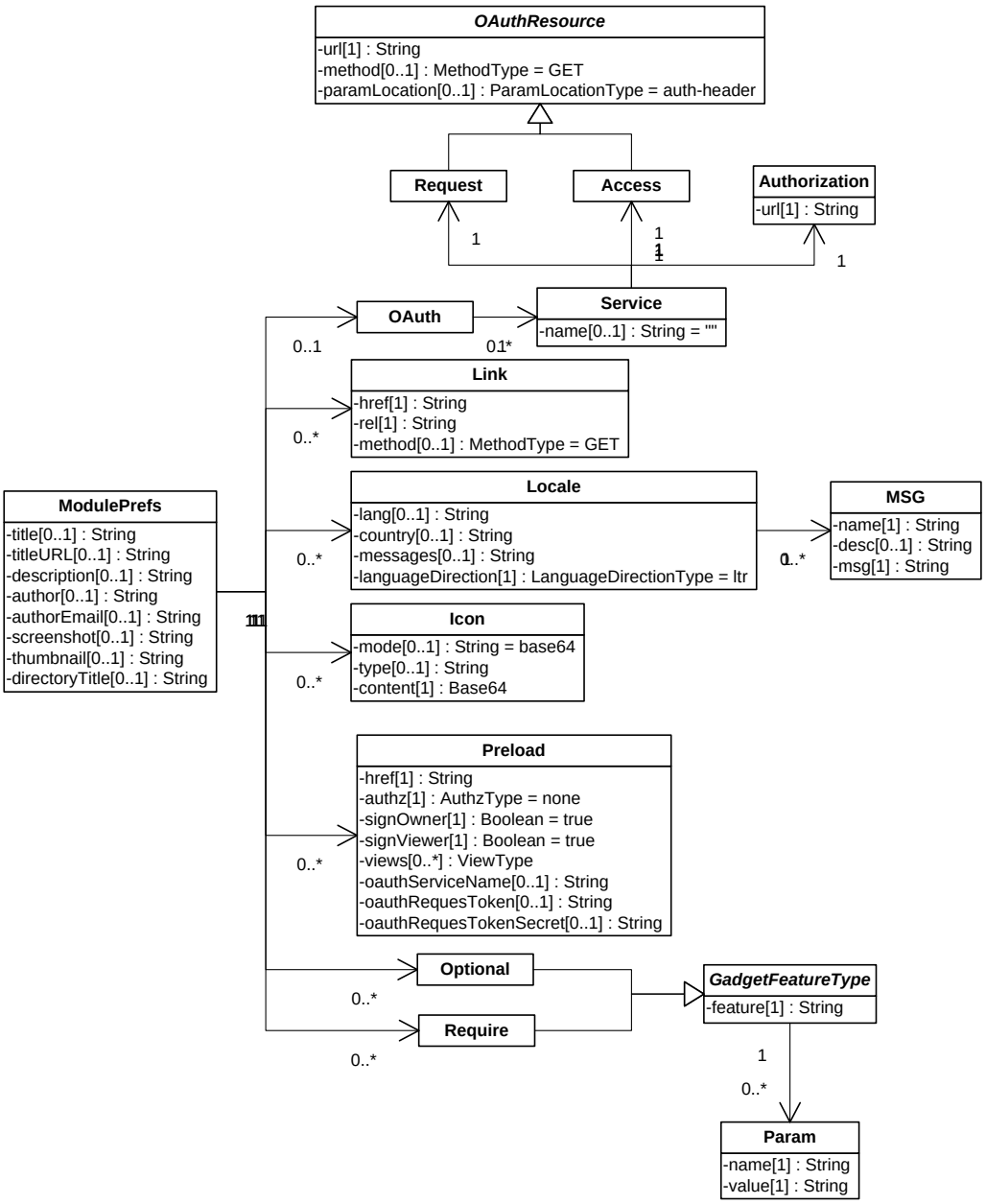


Figure 2.10: Google Gadgets - ModulePrefs

be embedded into web pages, too. In the latter case, they are officially termed "badges"[12] by Yahoo.

Picking up the discussion from Section 2.4 at a high level, we observe that both approaches (widgets/gadgets and portlets) tackle the same concepts. However, the widget/gadget approach allow for greater flexibility, due to advanced means for UIs definitions, mostly because of the functionality provided by HTML and CSS, in addition to the new Canvas element introduced in HTML 5 (Yahoo Widgets already use the `Canvas` element see Figure 2.11). From the run-time perspective both the

[12]http://widgets.yahoo.com/badges/

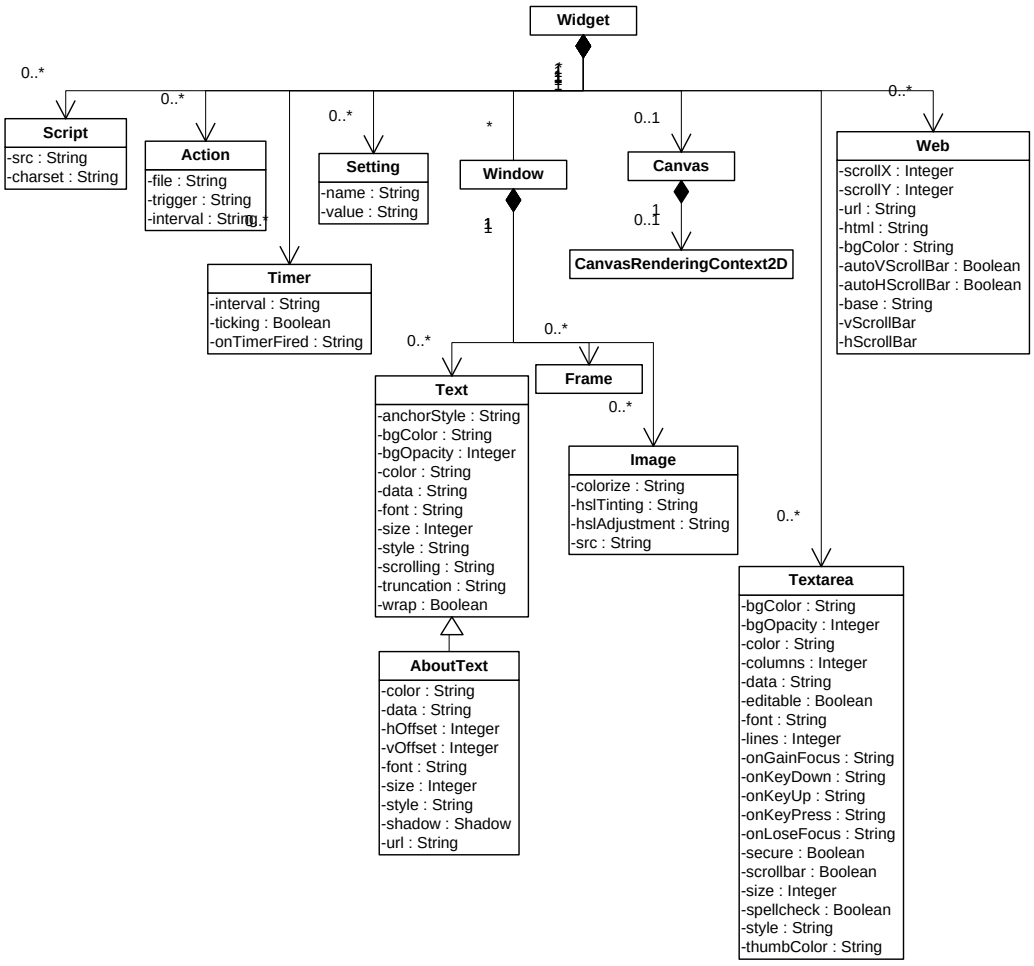


Figure 2.11: Yahoo Widgets Information Model 1

widget/gadget and portal approaches need some sort of a server that delivers the content.

2.5.2 Pipes Based Mashups

The concept of a pipe has been introduced by Doug McIlroy[13] in 1964, and refers to data aggregation in the context of macros: The output of a process becomes the input of another process, i.e., a pipe is a data processing pipeline [Hohpe et Woolf, 2003]. In the mashups context, pipes are mainly about data aggregation. Various sources of data (typically Atom or RSS feeds) are queried and aggregated to create a new data source (usually a new feed).

Yahoo Pipes

Pipes for mashups were pioneered by Yahoo with their Yahoo Pipes[14], which allows users to aggregate feeds and create data mashups using a visual editor. Pipes are created by combing several building

[13]http://cm.bell-labs.com/cm/cs/who/dmr/mdmpipe.pdf, http://www.princeton.edu/ hos/frs122/precis/mcilroy.htm
[14]http://pipes.yahoo.com/pipes/

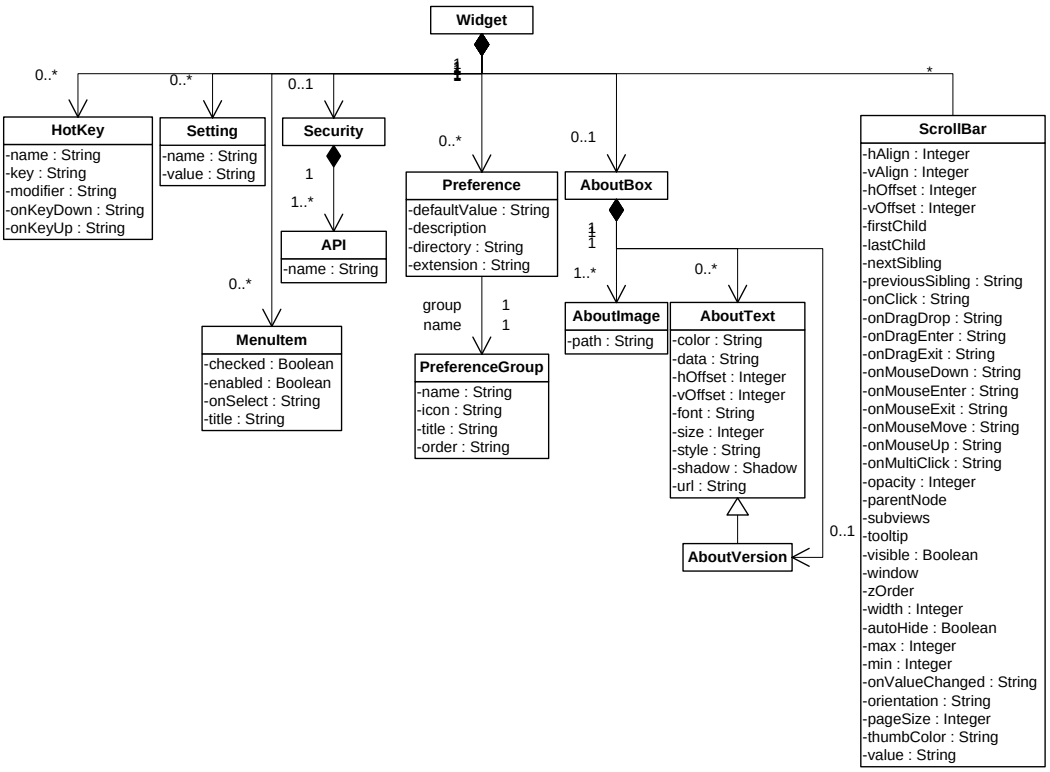


Figure 2.12: Yahoo Widgets Information Model 2

blocks, such as: `User Inputs`, `Sources`, `URL`, `Operators`, `Location`, and so forth.

A Yahoo pipe can have several inputs, but has only one output. The output is usually a feed (RSS or Atom). Another class of concepts relevant to the pipes approach is the *operator*. Some of the provided operators were greatly inspired from relational algebra such as sort, filter, or union, others are control flow operators such as loops. Special modules that deal with most common data types are also provided. Modules for `Numbers`, `Strings`, `Date` are defined. Modules for specific contexts, e.g., `Location`, have been taken into consideration, as well.

Semantic Web Pipes

Morbidoni et al. [Morbidoni *et al.*, 2007] define a

> semantic pipe as implementing a predefined workflow that, given a set of RDF sources (resolvable URLs), processes them by means of special purpose operators.

The model is not a fully fledged workflow but rather a simple construction kit that consists of linked operators.

Semantic Web Pipes deal with data aggregation as well as with processing the data in meaningful ways. Various building blocks are defined to process data sources, e.g., HTML, XML and JSON, and for processing data (mainly RDF data), i.e., operators (defined on top of the SPARQL query language) as well as user input blocks and conditional blocks. In addition, these pipes use an XML language behind the visual notations.

As a consequence, a semantic mashup in this context is basically a data mashup using RDF(S) as data model and SPARQL to query and process data. However, while in the basic case semantic pipes just aggregate semantic data, semantic mashups may have reasoning capabilities, and semantic web reasoners, such as KAON2[15], Pellet[16], Racer[17], Jena[18].

2.5.3 Hybrid

IBM Mashup Center

IBM Mashup Center has at its core a pipes-based technology, which has been improved with XPath constructs. In addition it provides also means to create widgets. These widgets use as data sources the feeds created with the corresponding pipes based tool.

Singh states in [Singh, 2008] that the IBM Mashup Center facilitates individuals to integrate and share information from diverse sources across organizational boundaries. "Data from multiple data sources can be merged, filtered, sorted, grouped, and transformed to create feed mashups". In addition, according to [IBM, 2008] the tool deals only with feed mashups i.e. a *feed mashup* "is a feed that you create by taking a source feed and applying operators and functions to filter and restructure source data".

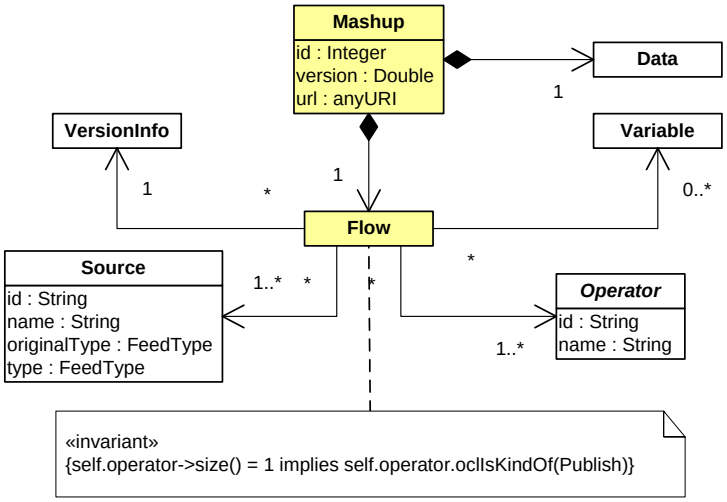


Figure 2.13: The Core Concepts of IBM Mashup Center

The general model of an IBM mashup is depicted in Figure 2.13. A mashup comprises a `Flow` and `Data`. While the most important concept is the `Flow`, the `Data` concept is used to store intermediary or output data. The `Flow` holds the whole logic of the mashup. It contains at least one `Operator`, the `Publish` operator, may have flow variable declarations (`Variable`) and has one or more data sources (`Source`). An IBM Mashup according to IBM view has a name and an `URI`. The URI is used to identify the feed created as a result of the mashup execution. Consequently, IBM mashups can be composed i.e. mashups can be integrated in other mashups, since their output is also a feed.

[15] http://kaon2.semanticweb.org/
[16] http://pellet.owldl.com/
[17] http://www.racer-systems.com/
[18] http://jena.sourceforge.net/

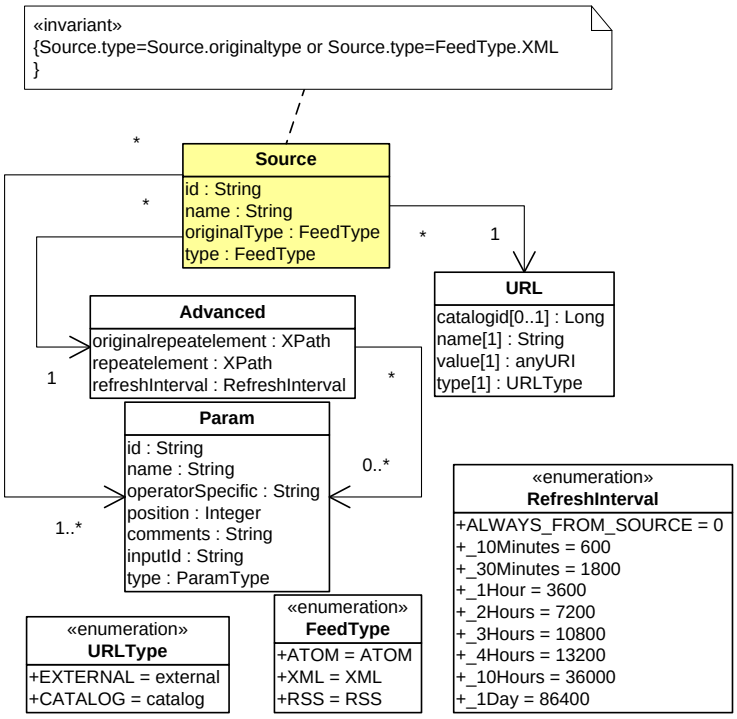


Figure 2.14: The Source Concept of IBM Mashup Center

The `Source` is depicted in Figure 2.14. The Source encodes both the declaration of the data source (its URI) but also its invocation. The URL expresses through its properties if it is an external feed or an internal one, belonging to the catalog. It also points towards the physical location of the feed. In addition, `Source` comprises a set of "advanced properties" (`Advanced`) covering the interpretation of the data: when a feed is accessed the system automatically finds the type of the feed and stores the type in the `originalType` attribute. However the user is allowed to declare how the system is going to interpret the feed data, by storing a specific data type in the `type` attribute. However, there is not too much freedom since the user defined `type` can be either the `originalType` (default) or `XML`. Finally, other properties such as `originalrepeatedelement` and `refreshInterval` encodes the repeating element of the feed and the frequency with which the feed should be retrieved.

Data sources can be parameterized declaring specific information on sources. Example 2.5.1 shows an excerpt from a source declaration in a mashup.

Example 2.5.1 (An IBM Mashup data source)**.**

```
<source xmlns="http://ibm.com/mashuphub" id="source598"
        name="Weather Mashup" originaltype="RSS" type="RSS" >
 <url catalogid="67" caption="Weather Mashup" type="catalog">
 http://weather.info/mashuphub/client/plugin/generate/entryid/19/pluginid/10
 </url>
 <advanced
   originalrepeatelement="/rss/channel/item"
   repeatelement="/rss/channel/item"
   refreshInterval="3600"/>
 <param id="source598_path0" name="domain name"
     operatorspecific="http" type="text">
     weather.info:9080
 </param>
```

```
</source>
```

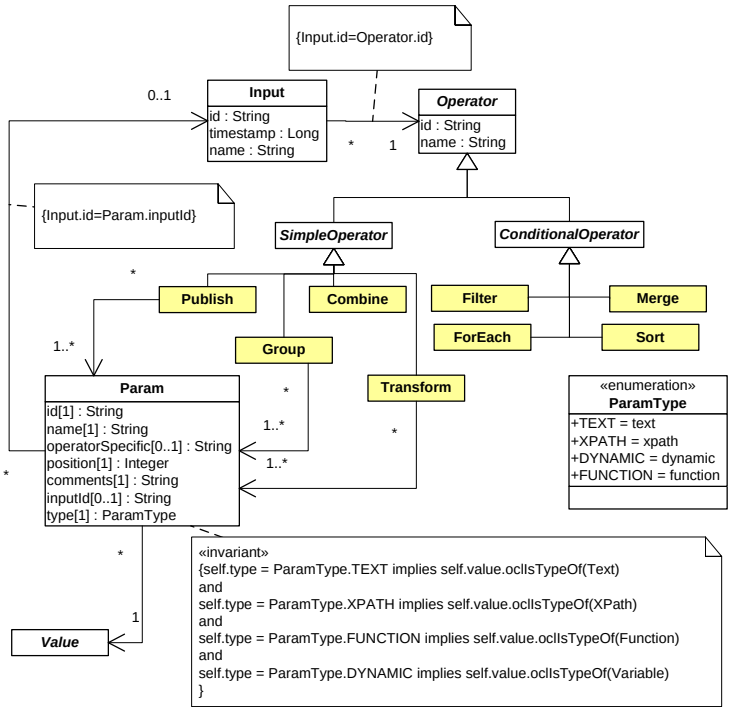


Figure 2.15: The IBM Mashup Operators

Figure 2.15 depicts a general view of the operators offered by the IBM mashup modeling environment. There are *simple operators* and *conditional operators*. Conditional operators take usage of conditional expressions. The operators are key constructs implementing the flow. They contain inputs (derived from data sources) and possible conditions on the data that has to be processed. Example 2.15 shows a simple `Sort` operator. Its execution results into creation of an ascending sorted list of cities. The mechanism of implementing the flow is built via ids i.e. the input `id` (e.g. `transform5284`) refers to the id of another mashup operator.

Example 2.5.2 (A simple IBM Mashup operator)**.**

```
<operator xmlns="http://ibm.com/mashuphub" id="sort6490" name="By City" type="sort">
 <input id="transform5284" name="cityAscending" timestamp="1212104921260"/>
 <condition id="cond4664" conjunction="yes">
  <param id="param6156" name="city" inputid="transform5284" type="xpath">
   ./content/row/ns3:city/text()[1]
  </param>
  <param id="param9523" name="sort" type="text" >
   0
  </param>
 </condition>
</operator>
```

JackBe Presto Wires

This section presents the general model behind the JackBe Presto Wires composer [Presto, 2009]. JackBe developed an enterprise mashup platform, which enables application developers and end users

to experience self-service data access, and situational integration to make more effective decisions and independently complete daily tasks [Presto, 2009].

Figure 2.16 depicts the general UML model of the JackBe mashups.

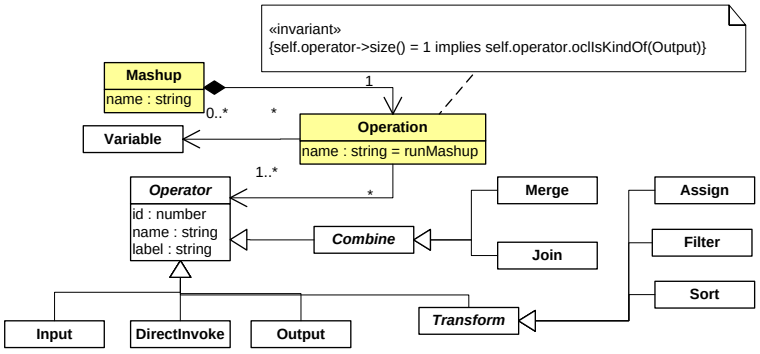


Figure 2.16: The Core Concepts of JackBe Presto Wires

It is straightforward to see the strong similarities with the IBM mashups. However while IBM defines a `flow`, JackBe defines an `operation`, unique in the scope of a mashup. An operation has a `name`. Such a JackBe operation may have variable declarations which are global as in the IBM Mashups case. The mashup `operation` contains one or more `operators`. JackBe operators include `Input`, `DirectInvoke`, `Output`, and also operators concerning data transformation and data aggregation. Operators may have inputs but they must have outputs (possibly stored in a `Variable`(s)).

Therefore, opposed to IBM mashups which defines this behavior implicitly, JackBe distinguish between the input (`Input`) and its invocation (`DirectInvoke`) as well as its result (`Output`).

This distinction shows the strong influence and orientation towards web services (SOAP and REST) paradigms (see WSDL [Chinnici *et al.*, 2007], and WADL [Hadley, 2006]) while IBM mashups are much more oriented to proceses/flows.

Example 2.5.3 depicts a simple excerpt of a `join` operator in JackBe. Unfortunately their XML language does not really get advantage of XML markup - they widely use attributes instead of elements (see, for example, the `joincondition` or `inputvariables`).

Example 2.5.3 (Simple Operator in JackBe)**.**

```
<join label="Join" description="Simpel"
   joincondition="$Direct_Invoke_60_out=$Direct_Invoke_63_out and
                  $Direct_Invoke_60_out=$Direct_Invoke_63_out"
   outputvariable="Join_58_out" name="Join_58"
   inputvariables="Direct_Invoke_60_out, Direct_Invoke_63_out" id="58">
  <select>
   <Direct_Invoke_60_Direct_Invoke_63/>
  </select>
</join>
```

Enterprise Mashup Markup Language

JackBe Presto provides also a (Domain Specific Language) DSL based approach: Enterprise Mashup Markup Language (EMML) [Alliance, 2009, Presto, 2009]. Initiated by JackBe[19], EMML is now provided by the Open Mashup Alliance[20] to which belong major companies, e.g., JackBe, Kapow Technologies, Intel, Adobe, HP, working on a common standard for mashup solutions.

[19] http://www.jackbe.com/

[20] http://openmashup.org/

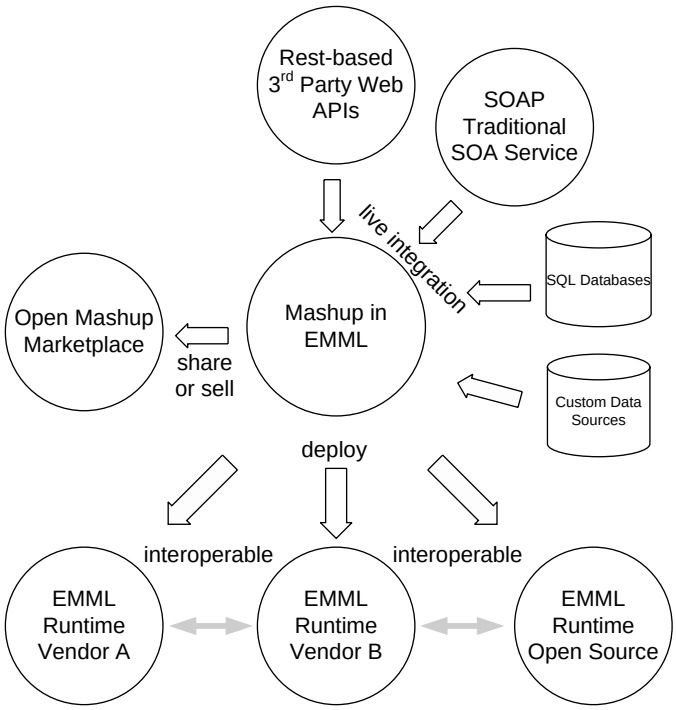


Figure 2.17: EMML, Dion Hinchcliffe - http://blogs.zdnet.com/Hinchcliffe/

Figure 2.17 depicts, at a high level, the capabilities of EMML. The languages has been developed to allow the definition of almost all the concepts used in the approaches described in previous sections. As such, data flows can be defined using operators similar to Yahoo Pipes: API based aggregation, Web clipping, and JavaScript embedding. In addition, the language provides constructs that deal with SOAP based Web services, also BPEL similar constructs are provided. Specific constructs for accessing enterprise data sources are also provided. Even though JackBe provides different environments for developing such description files, they are very difficult to address.

The JackBe Mashup server allows the user to publish created mashups as Web services, in turn.

mashArt

mashArt is an approach that comprises "a unified component model and a universal, event-based composition model, both able to abstract from low-level implementation details and technology specifics" and aims at "empowering users with easy-to-use and flexible abstractions and techniques to create and manage composite web applications" [Daniel *et al.*, 2009]. This approach addresses the mashups concept from a higher abstraction perspective.

The universal component model allows the modeling of UI components, application components, e.g., services with an API, and data components, e.g., feeds or access to XML and relational data. The component model is based on four abstractions:

- **state:** represented by a set of name-value pairs; addresses changes that are relevant to other components;
- **events:** communicate state changes and other information as name-value pairs to the composition

environment; a subscription mechanism is in place;

- **operations:** invoked as result of events; often represent state change requests;
- **properties:** arbitrary component setup information.

Combination of building blocks, as well as exposing the result as a component is done by means of the universal composition model. The envisioned composition model deals with both stateful and stateless components and UI composition. The authors argue that the model has features from event-based composition as well as from flow-based composition, cf., [Daniel *et al.*, 2009].

2.5.4 Domain Specific Languages for Mashups

Domain specific languages (DSLs) are built on top of hosting languages and abstract from technical details, tackling a particular problem domain. Through moving the design of a system into the problem domain, rather than solving it on a technical platform, DSLs can reduce complexity and increase the efficiency of system development.

MashQL.

MashQL [Jarrar et Dikaiakos, 2008] is a domain specific language, that mainly deals with data mashups. The language is influenced by Yahoo Pipes. However, an important aspect is that Jarrar and Dikaiakos regard the Internet as a database, where each source is seen as a table, and a mashup is a query on these tables. Moreover, they assume that the Web data sources are represented in RDF and that SPARQL is the query language.

MashQL is a *query-by-diagram* language. The authors argue that users can *formulate queries over a data source without any prior knowledge about its schema* [Jarrar et Dikaiakos, 2008]; in addition, no RDF or SPARQL knowledge is required to get started. Four constructs are provided to issue queries over such web data:

- **Query-by-Form:** users fill in and submit a form; form fields are seen as query variables;
- **Query-by-Example:** queries are formulated as filling a table; names of the queried relations and fields are selected first;
- **Conceptual-Query-Languages:** users select some concepts from a given conceptual diagram and their relation is automatically translated into SQL queries;
- **Query-by-Filter:** permit/block items according to a certain set of conditions.

Mashlets.

The notion of a Mashlet has been introduced in [Abiteboul *et al.*, 2008]: Abiteboul et al. define a mashup as a dynamic network of interacting mashlets. A mashlet may query data sources, import other mashlets, use external Web services and specify complex interaction patterns between its components. Mashlets are based on object databases [Cattell, 1994] and on active databases [Widom et Ceri, 1996].

The state of a mashlet is maintained and represented by a set of relations:

- **internal relations:** store internal data;
- **input/output relations:** interaction with other mashlets or with users;

- **service relations (bidding pattern):** capture web services that a mashlet imports or exports.

Mashlet components, as well as interaction between them, can be defined statically or dynamically at runtime. A mashlet includes in general five relations: Inputs, Outputs, Mashlets, MashletAPIs and Rules.

Swashup DSL.

Swashup is another DSL introduced by Maximilien et al. [Maximilien *et al.*, 2007] and addresses mashups with the help of three main components: (1) data and mediation, (2) service APIs and (3) means to generate Web applications with customized UIs. The following concepts are Swashup constituents:

- **data:** describes data elements used in a service. Such an element corresponds to an XML schema type. Each data element has a name and a set of attributes;
- **api:** provides a complete description of a web service interface, including a service's API (operation names, parameters, and data types);
- **mediation:** describes how data elements have to be mapped to each other, including possible transformations;
- **service:** binds a service **api** to a concrete service;
- **recipe:** constitutes a collection of services and mashups; includes also views for each of the mashup wiring:
  - mashup: composition of one or multiple services; comprises a collection of wiring declarations; each mashup is translated into a composed service that can be further used by other mashups;
  - mediate: invokes a **mediation** declaration;
  - wiring: includes a protocol definition and operations;
  - step: one atomic step in a protocol mediation, a step is invoked by the step's name as a method call;
  - tag: annotate terms.

2.6 Discussion

Based on the previous extensive considerations on trends that have influenced the creation of mashups as well as based on the research directions identified by the EU FP8 Expert Group working on Services in the Future Internet [21] in the next paragraphs we are going to make a series of assertions and considerations about the reasons for which we believe the current approaches did not really achieved the desired outcome; we extract a minimal common model (see Figure 2.18) for mashups based on mashups approaches presented earlier; we enumerate the requirements that we consider a mashup

[21] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, retrieved 13 January 2014
http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, retrieved 13 January 2014

approach should comply with. Our approach which we are going to develop and present in this thesis will comply with the identified requirements.

Although mashups have been initially thought and developed only from a technical perspective, mainly through hacks, we believe that the conceptual side of the problem needs to be taken into account as well.

Mashup development is a promising End User Development (EUD) application area as argued in [Grammel et Storey, 2008]. By using services that can be accessed through the web as the underlying platform the development effort is shifted from traditional programming. Hence the challenges faced by mashup tools designers include the need for defining a high level way for describing and combining computation, integration logic and abstractions to represent Web widgets, Web Services, Web data sources [Aghaee *et al.*, 2012]. Nonetheless these goals have not been yet achieved.

A large number of mashups tools have been developed both in the academia and in industry, however only a few have succeeded. Many of these approaches even though have been developed by big companies such as Microsoft or Google have been discontinued, i.e. Microsoft Popfly, Google Mashup Editor. Others such as JackBe Presto or Serena Mashup Composer are still in use. We believe that there is a big interest for such tools both in the academia and industry and new frameworks are under development, i.e. [Aghaee *et al.*, 2013]. In the same time we argue that the current approaches did not really achieved the desired outcome because were either too technical or too scientific, and they lacked a proper integration between the technical perspective and the conceptual perspective. In this way these approaches lost on the way the fundamental target - the end-user. We believe that this state arises from the fact that currently software development process in Software Engineering is mainly focused around the structure of the software being built and the interactions between its components. When in fact the focus should be on the end-users.

Albeit client mashups as argued in [Bioernstad et Pautasso, 2007] are real mashup, almost all of the approaches we have discussed throughout this chapter are server-side mashups. We believe that user-centric approaches for mashups should be client based mashups. End-users should not be required to install and configure and maintain servers. End-users however should be provided with tools that are general enough to address most of the technologies enumerated here in a unified and holistic way. These tools should hide as much as possible the engineering side and the technologies. Moreover these tools should be the same no matter the type of device that is being used. We assert that such tools can be achieved with the advance of HTML5 and JavaScript. We believe that a browser based solution can comply with all these aspects. Other researchers are agreeing with us on this matter. One could see for instance [Aghaee et Pautasso, 2010].

Publishing and content syndication are important aspects in mashups' life cycle. Based on the discussed approaches it seems that data is the most important asset for mashups and that data is the glue that links together mashed sites. The most common formats used are RSS [RSS, 2009] and ATOM [Gregorio et de hOra, 2007]. Although publishing formats must be understood and known very well by the creator, the general trend is to use XPath [Berglund *et al.*, 2007] expressions to query and or modify data. The use of XPath is embraced both by IBM Mashup Center and JackBe Presto. Data consumed by mashups gets mixed, remixed and refashioned in a multitude of views imagined by the end user – the creator. Yahoo Pipes and Microsoft Popfly follows the same architectural design. Figure 2.18 shows a minimal common model of mashups based on previous comparison and according with others observations. Since discussed models use different vocabularies, Table 2.4 depicts a vocabulary mapping.

According to Figure 2.18 a user has to define a flow (choreography [Weske, 2007]) that takes place inside of a "main" activity exposed by the mashup service. The mashup's flow is composed of *activities*. There is a general rule that, flow elements have at least one ativity (i.e. `Publish`). However, the

Minimal Model	IBM Mashup Center	JackBe Presto Wires
Source	Source	Input
Flow	Flow	Operation
Activity	Operator	Operator
Output	implicit	Output
Input	Input	Input
Publish	Publish	Output

Table 2.4: Terminology

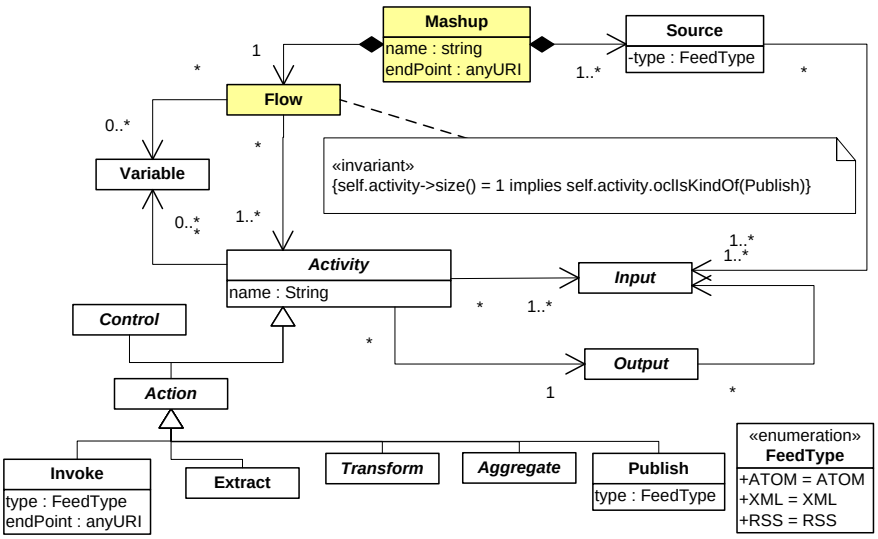


Figure 2.18: The Core Concepts of Mashups

mashup flow, opposed with other workflow languages such as YAWL [van der Aalst et ter Hofstede, 2005] or BPMN [Weske, 2007] it is strongly data oriented. As such its activities are classified into two major categories: those who produce state changes i.e. *actions* (*data transformation actions* such as `Assign`, `Sort`, `Group`, and `Filter`, *data aggregation actions* such as `Join`, and `Merge`), and *control flow operators* (such as `Sequence`, `Parallel`, `Loop`, `Conditional`).

We argue that mashups tools should deal not only with data in RSS or ATOM format but also regular data. Moreover behavior (events, actions) should be taken into account as well.

We also assert that it is possible to address semantics without the need to annotate and recopy the web contents, as it has been done in the case of DBpedia. (we presented a larger discussion on the mater in Section 2.2) We believe that this can be achieved by using a similar approach as the one used to automatically extract and annotate the contents. We will explain this approach in later chapters.

We arrive to the list of requirements with which we believe an end-user oriented mashup system should comply with:

R1 such a system should allow evolution, sharing and distribution; end-users should be allowed through direct input to update / adapt the application;

R2 such a system should not be domain specific, and should allow a wide range of use-cases;

R3 in a such a system the end-user should be the coordinator of how the system works; hence the system should provide a new approach for describing behavior of composite systems (humans + services) [22];

R4 such a system should support both skilled developers as well as novice users;

R5 such a system should focus on the end-user and not on the system itself. The system should be hidden from the end-user as much as possible by providing the right level of representation such that a problem representation could be automatically translated into the core concepts of the underlying programming language in which the overall system is implemented;

R6 such a system should allow on demand development using the web as a platform or web as open application execution environment: build upon existing ideas, sites, applications [23];

R7 such a system should be compliant with SOA principles of: loose coupling, reusability, discoverability, composability;

R8 such a system should allow decentralized and delocalized execution of software / components [24];

R9 such systems should allow simultaneous build time, run time development and experiment [25].

Having this discussion as starting point we will present in the following chapter our approach as well as the conceptual model which we propose to tackle the issues underlined in this section.

[22] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, retrieved 13 January 2014
[23] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, retrieved 13 January 2014
[24] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, retrieved 13 January 2014
[25] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, retrieved 13 January 2014

Chapter 3

A User-centric approach. Conceptual Model

We have emphasized in the Discussion section (Section 2.6) of Chapter 2 a series of characteristics and requirements that we believe user-centric mashups should comply with. Thus such systems should be end-user oriented. Such systems should provide a new approach for describing behavior of composite systems (humans + services) R3. We argue that such systems are intelligent systems being able to interact with the end-user according with an agreed beforehand plan, supporting evolution, sharing and distribution R1. Hence these systems are two layer systems: one high level layer, that deals with the problem at a conceptual and semantic level (the agreed on plan) and one low level layer that deals with the internals of the system and low level technologies i.e. direct access to services, etc. The low level layer should be hidden as much as possible from the end-user R5.

The aspects (also depicted in Figure 3.1) that drive the development of our approach are: end-user oriented or user-centric; humans and system interacting with each other; plan; two-layer system; intelligent system. These aspects have been previously discussed in the research literature, however almost always in a disconnect manner with almost no interaction with each other. In addition different terminology, according to the research directions where it has been studied, has been used to identify actually the same concept.

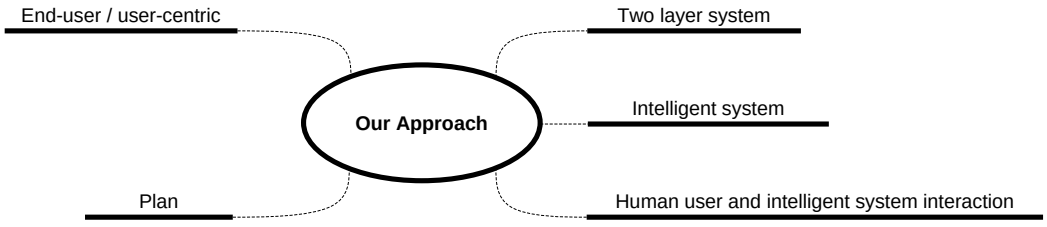


Figure 3.1: Aspects that drive our approach

This chapter describes the approach we propose and the conceptual model we created in relationship with our approach.

3.1 Our proposal - A User-centric approach

Because at the heart of the mashup phenomenon are the users and the innovation that users bring [Cappiello *et al.*, 2011] we start our discussion about the aspects driving our approach with the ***end-user*** perspective.

3.1.1 End-user / user-centric

End-user development has been defined in [Lieberman *et al.*, 2006] as a

> set of methods, techniques, and tools that allow users of software systems, who are acting as nonprofessionals software developers, at some point to create, modify, or extend a software artifact.

End-user software engineering research is interdisciplinary [Lieberman *et al.*, 2006] involving ideas from: computer science, software engineering, human-computer interaction, education, psychology and other disciplines.

The difference between professional programmers and end-user programmers are their goals [Ko *et al.*, 2011]. Application developers can no longer anticipate all the needs of end-users. In addition professionals are required to develop software that can be maintained over time as opposed to end-users who write software mainly for personal use and to support a particular goal. End-users do this by employing pre-existing computer applications. Often to make these application truly useful end-users need to adapt these applications to their specific needs [Repenning et Ioannidou, 2006]. These adaptations can take many forms, ranging from simple forms i.e. application preferences to more complex issues such as email filtering rules. There is an increasing need to be able to address these complex forms of adaptations [Repenning et Ioannidou, 2006]. For instance browsers are used to access fast growing information spaces. In this context only end-users of applications can decide what to do with this large amount of information.

In consequence (see Figure 3.2) the *end-users* whom we want to support, are most of them, *not professional developers*, who lack technical skills, and don't have the necessary knowledge to write software programs, according to technical specifications (specifications for Web Services, APIs, REST etc). End-users have a variety of *goals*. These goals are achieved by *creating* new applications, via mashing up existing applications, or software artifacts, by *modifying* or *adapting* existing applications.

3.1.2 Two layer system

Now in order to allow end-users, who are not professional programmers to program, adapt existing applications or software artifacts according to their needs the technical layer needs to be hidden as much as possible, otherwise they will not be able to do it.

To achieve this separation of concerns and thus hide the technical layer we argue that a ***two layer system*** is required. The high level layer should provide the means to allow both human users and the system to understand each other using a common set of concepts and following a beforehand agreed on ***plan***. The low level layer on the other hand should be hidden from the end-user and should be accessed directly by the system according to what has been agreed upon in the ***plan***. A professional developer should also be allowed access to this layer.

Software Engineering (SE) is the field of study that is concerned with all the aspects related to the design and development of software systems. Two of these aspects: Requirements engineering and

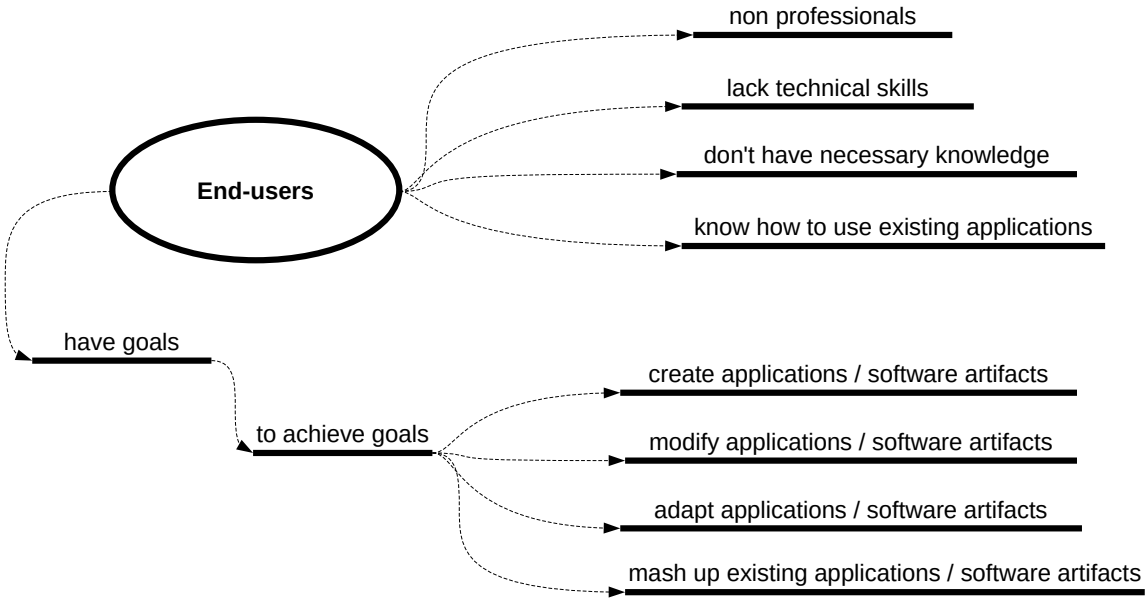


Figure 3.2: End-users related aspects

system design are of interest for our approach because system design targets the internal behavior of the system while requirements are external, concerning the world.

The phase of gathering requirements precedes system design. Unfortunately in most of the situations the final product does not actually comply with the end-user expectations for various reasons: i.e. bad communication, different understanding of concepts and situations, etc. [Tognazzini, 1992, Nardi, 1993]. An end-user perceives and understands a software system through the user interface (UI). Based on the UI, which is supposed to be an accurate and complete representation of the system, the end-user builds its own understanding of the environment consisting of concepts, with which it is working, and associated behavior [Clark et Sasse, 1997]. Haplessly in many situations the end-user's understanding is very different than the understanding and the message, developers actually tried to convey [Tognazzini, 1992].

A requirement in Requirements Engineering (RE), as stated in [Pohl, 2010],

> defines both needs and goals of users, and conditions and properties of the system to be developed, that result for example, from organizational needs, laws, or standards.

In RE a goal is a stakeholder's intention with regard to the objectives, properties or use of the system [Pohl, 2010]. Requirements are said to define *what* should be developed while system design defines *how* the system should be developed [Pohl, 2010].

In consequence we debate that RE stands for the high level layer and system design stands for the low level layer. And therefore we believe that both RE and system design need to be unified when dealing with proper intelligent user-centric systems (see Figure 3.3).

3.1.3 Plan

Requirements Engineering approaches propose also the use of scenarios, which are concrete positive or negative examples of satisfying or failing to satisfy a goal or a set of goals [Pohl, 2010]. Such scenarios

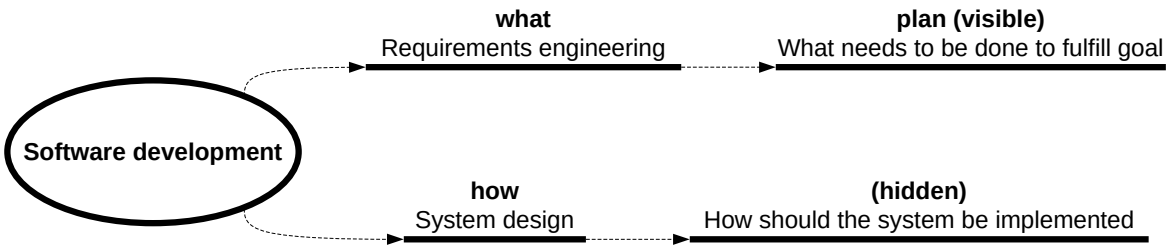


Figure 3.3: Software development aspects

for our approach are the ***plans*** that end-users create. Software requirements are usually expressed in natural language. However natural language can not be an option here. We need a structured and well defined set of concepts to express end-user plans, such that an intelligent system can understand, reason and use that plan, in its interaction with the environment and the end-user. Because this plan needs to be from the perspective of end-user we looked first to research domains such as Human Computer Interaction (HCI), conceptual design and cognitive psychology.

In these domains such a user defined plan is assimilated to the notion of a mental model [Clark et Sasse, 1997]. Such plans, in cognitive psychology have been conceptualized as "representations in episodic memory of situations, acts or events spoken or thought about, observed or participated in by human actors" [van Dijk, 1997]. According to [McDaniel, 2003] such a plan consist of several parts:

- *an image*: if the mental model refers to a physical object then the model should contain a simplified image of the object;
- *a script*: if the mental model refers to a process, it should contain a description of that process;
- *a set of related mental models*: mental models can be composed of other mental models;
- *a controlled vocabulary*: each mental models has a set of key definitions and variants;
- *a set of assumptions*: allow users to predict behavior.

Translating this description into the Web context, an end-user plan contains a set of concepts, with which the end-user works with in order to fulfill a goal, their relationship with each other, the context; and means to express behavior in the forms of processes and rules. Hence for our approach the user has a plan of *what* needs to be done in order *to achieve* a *goal*. *Needs to be done* here means the *interaction (behavior)* that a user needs to exhibit with the application (applications) in order to fulfill the goal. Moreover the user is expecting a particular answer from the system as a response to the actions he/she, the user, performs. User behavior is imposed (influenced) by the *context (environment)*.

Traditionally context has been perceived in computer science community as a matter of location and identity, see for instance [Dey et Abowd, 1999b]. However interaction and problems concerning interaction require more than just the environmental context (location, identity) used traditionally in context-aware systems [Grudin, 2001]. As such lately the notion of context has been considered not simply as state but as part of a process in which users are to be involved [Coutaz *et al.*, 2005]. Fischer, however gives a definition that takes into account the human-centered computational environments. Context is defined in [Fischer, 2012] as being *the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way to the 'right' person*.

Context greatly influences the way humans or machines act, the way they report themselves to situations and things; furthermore any change in context, causes a transformation in the experience that is going to be lived, sensed [Bolchini *et al.*, 2007]. Many psychological studies have shown that when humans act, and especially when humans interact, they consciously and unconsciously attend to context of many types as stated in [Grudin, 2001]. Nardi underlines this aspect clearly in [Nardi, 1993] stating that

> we have only scratched the surface of what would be possible if end users could freely program their own applications... As has been shown time and again, no matter how much designers and programmers try to anticipate and provide for what users will need, the effort always falls short because it is impossible to know in advance what may be needed... End users should have the ability to create customizations, extensions and applications....

For the approach that we envision and design in this thesis, the user creates a *plan* that is shared (given) to the system. This plan contains a description of the *context(s)* and the *behavior* that both the human user and the system need to perform in relationship with the context(s) as we introduced it in [Pascalau, 2011a]. The plan explains how the system should react in response to the actions that the human user performs, or how the system should react in response to changes that appear in the environment (context(s)). In this way both the human user and the system will follow and will share the same understanding, the same plan. Petrelli *et al.* emphasize this [Petrelli *et al.*, 2000] by stating that the main objective of context should be to make technology invisible for the user, hence the focus will not be anymore on how to use the technology in a proper way but the focus will be once again on resolving our activities and achieving our goals in daily business activities and not only. To achieve this objective will require a completely and reliable world representation; a shared understanding of concepts [Petrelli *et al.*, 2000]. For our design we adhere to the ideas discussed in [Bolchini *et al.*, 2007] and we argue that the world representation needs to be from an individual's perspective. Such an individual representation of the world needs to be a subset of the entire world that a system can understand.

3.1.4 Intelligent system and human user and system interacting with each other

The last two aspects that we identified concerning our approach are: ***intelligent system*** able to ***interact with the human user*** according to the agreed plan.

Artificial intelligence and Cognitive science are two of the research topics that address the problem of intelligence and reasoning in machines and software. However these topics are developed most of the time separately although they work with concepts that often overlap.

Russell and Norvig define in [Russell et Norvig, 2009] an agent as anything that can perceive its environment through sensors and then act in the environment through effectors. Agents are entities that function continuously and autonomously in an environment in which other processes are running and in which exist also other agents [Shoham, 1993]. Based on this definition humans are agents, since humans have eyes, ears and other organs that act as sensors and have hands, legs, mouth and other parts of the body that act as effectors.

We assert that our system can be assimilated to the notion of an agent. A hybrid agent: a combination between a reactive agent, a deductive agent and also proactive agent.

Cognition implies the ability to reason about the environment, about how things might change over time, and how one should act according to this information [Vernon *et al.*, 2007]. Vernon [Vernon *et al.*, 2007] continues and states that cognition can be viewed

> as the process by which the system achieves robust adaptive, anticipatory, autonomous behavior, entailing embodied perception and action.

The basic concepts of intelligent systems - representation, knowledge, symbols, and search - apply both to humans and machines [Newell, 1994]. However human beings are very unique. Nevell argues that a single system (mind) produces all aspects of behavior (see [Newell, 1994]). Even though the mind has different components, these components mesh together to produce behavior. Behavior is flexible as a function of the environment [Newell, 1994]. Both a behaving organism as well as the environment behave through time with a series of interactions between them. The behaving organism takes actions as a function of the environment, hence if the environment is different an organism can behave differently even though is performing the same actions. Behavior is assumed to be governed by the rationality principle [Pirolli, 1999]. Thus if the user knows that one of his or hers actions will lead to a situation which is compliant with the goal, then he or she will intend to perform that action.

Previous paragraph completes our discussion about the notion of a ***plan*** by confirming that behavior is influenced by the environment; that is a characteristic of an intelligent system and that it comprises actions which are performed as response to perceived events in relationships with existing knowledge.

In addition we argue that our approach complies also with the notion of an intelligent system as described by Newell in [Newell, 1994] as it comprise the basic concepts of intelligent systems: representation, knowledge, symbols and search.

3.1.5 The approach

We link together the aspects we just discussed and we resume our approach in Figure 3.4. The approach we have introduced in this section proposes a new programming model through a composite system where human user and system interacting with each other. The interaction is via environment and is according to a predefined plan. This plan is created by the human user, hence the human is the coordinator of how the system works. Both the intelligent system and the human user follow the same plan. Such a plan serves to achieve a particular goal and it has been created taken into account the context(s). Interaction is perceived via changes that appear in the environment.

We believe that such an approach as discussed here is compliant with the meta-design framework, which is a conceptual framework defining and creating social and technical infrastructures in which new forms of collaborative design can take place [Fischer et Giaccardi, 2004]. Meta-design originates in human computer interaction field and tackles end-user development. The approach, we have introduced, in accordance with meta-design concerns adaptable systems where: users change the functionality of the system; users extend knowledge; user are controlling how the system works.

3.2 Conceptual Model

We enunciated earlier that the proper way to define our conceptual model is by means of ontologies [Guarino, 1998]. Instances of this model will represent the *plan* that a user is going to provide to the system.

Uschold and Jasper argue in [Uschold et Jasper, 1999] that though an "ontology can take a variety of forms, it will include a vocabulary of terms, and some specification of their meaning". Definition of concepts and their relationships impose a structure on the domain and constrain terms interpretation. As UML is considered to be the *de facto* standard modeling language [Guizzardi, 2005] we are also using it to formalize our conceptual framework. We will discuss one by one the main elements of our framework.

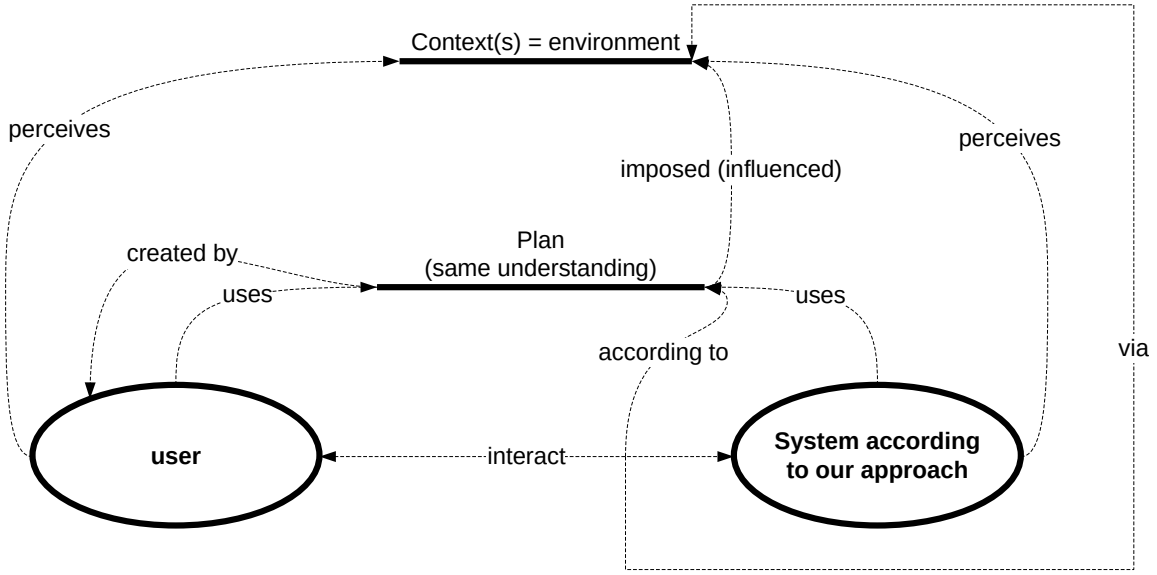


Figure 3.4: Our approach

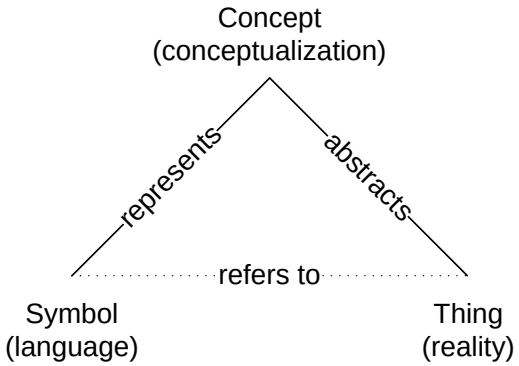


Figure 3.5: Ullmann's Triangle depicts the relationships between *things* in reality, the *conceptualization* of things and their *symbolic* representation. [Ullmann, 1972]

3.2.1 Concept

Figure 3.5 depicts Ullmann's triangle [Ullmann, 1972] which explains the relationship between *things* in reality, their *conceptualization* and their *representation*. The relationship should be read as follows: A *Symbol* represents a *Concept*. A *Concept* abstracts *Thing*, and a *Symbol* refers to a *Thing*.

Although many of the ontological approaches (see for instance OWL [Group, 2009]) use as the upper level entity the *thing* notion for our conceptual framework the most general unit of knowledge is the *concept* notion. Moreover the OMG specification for Semantics of Business Vocabulary and Business Rules Specification (SBVR) [OMG, 2008] uses as top entity the *concept* notion as well.

Definition 1 (Concept)**.** Unit of knowledge created by a unique combination of characteristics [OMG, 2008].

A concept has a name and a set of properties. It is a subclass of `uml::Class` entity. It can

be identified either by its name or by the set (or a subset) of properties that define it. In software engineering when dealing with typed languages, entities are recognized by their types (class name). The other way around is based on a set of characteristics. Take for instance a *car*. Stating the concept's name "car" someone will be able to tell you the characteristics of a car, that it has for wheels, that it has an engine etc. Nevertheless stating the characteristics of the concept that it has 4 wheels and an engine, the answer will be a *car*. In our approach both perspectives are taken into account.

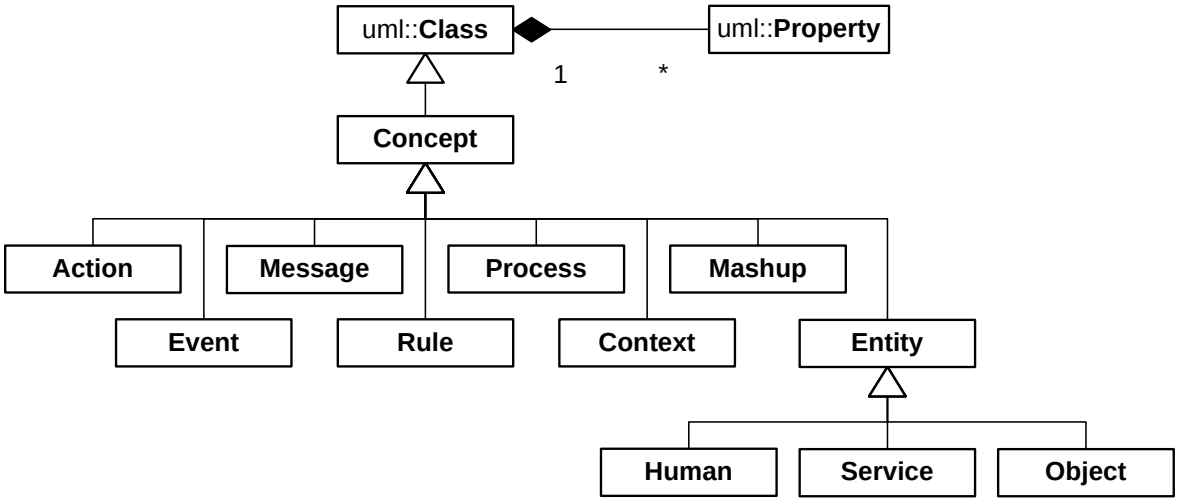


Figure 3.6: Concepts

Based on Definition 1 a concept is a unit of knowledge created by a unique combination of characteristics. Thus as depicted in Figure 3.6 every element of the framework is a concept. In this way the reasoning process can involve any of the concepts defined in a unified way.

Recall the conferences calendar use case we have introduced in Chapter 1. The goal is to automatically store events that are announced in the DbWorld mailing list in a Google calendar. Without an approach similar to the one described in this thesis, to achieve this goal of automatically storing events in a Google calendar requires two tabs open in the browser, requires going back and froth between these two tabs and copying and pasting each piece of information manually for each event. Figure 3.7 depicts the Google calendar service - the *save event* facet.

Google calendar service is an instance of the `Service` concept. Based on Definition 1 a concept has characteristics. Hence the characteristic of interest for us is the `URL` of the service: *https://www.google.com/calendar/*. Another concept of interest for our use case is the existence of the button `Save`. Empirically from an end-user perspective to identify the save button as being the save button requires that the button is a `button` and has the text content `Save`. A real representation, as a DOM tree, of the save button according to the current google template, inside a browser, is depicted in Example 3.2.1. Thus in order to identify the save button, the DOM tree representation of the Google calendar service must contain a concept that has the following characteristics: `type` with value `div`, `class` with value `goog-inline-block goog-imageless-button`, `role` with value `button` and one child that has as value another concept. This second concept has the following characteristics: `type` with the value `div`, `class` with the value `goog-imageless-button-content` and `textContent` with the value `Save`.

Example 3.2.1 (Google Calendar Save Button DOM representation)**.**

```
<div class="goog-inline-block goog-imageless-button" role="button"
```

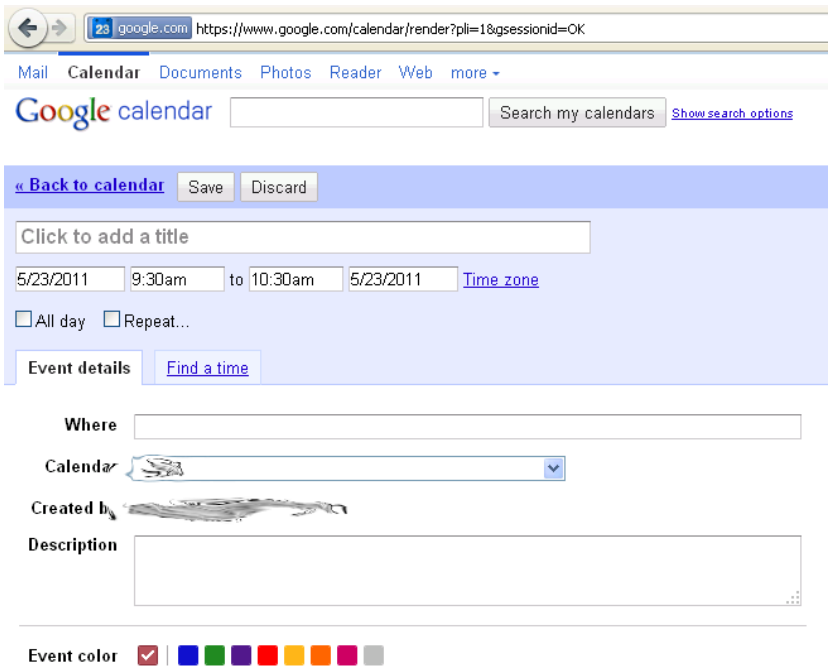


Figure 3.7: Google calendar service - save event facet

```
style="-moz-user-select: none;" tabindex="0">
<div class="goog-imageless-button-content">
Save
</div>
</div>
```

In a similar way all the other necessary concepts for this use case can be identified.

3.2.2 Context

Context greatly influences the way humans or machines act, the way they report themselves to situations and things; furthermore any change in context, causes a transformation in the experience that is going to be lived, sensed [Bolchini *et al.*, 2007].

Although studied intensively there is no one accepted definition for the context notion. Dey in [Dey et Abowd, 1999a] defines context as

> 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 applications themselves.

Context is under permanent change, is episodic, personal and hence subjective interpretations and experiences of the communicative context [van Dijk, 1997],[Dey et Abowd, 1999a]. Analyti et al. discuss in [Analyti *et al.*, 2007] a general framework to harness the notion of context in conceptual modeling. A full mathematical apparatus has been defined to tackle issues such as containment and relationships between contexts. According to them "context in an information base can be seen as a higher-order conceptual entity that groups together other conceptual entities on which we want to focus" [Analyti *et al.*, 2007]. More precisely a context is a set of objects within which each object is associated with a set of names.

For our conceptual framework the notion of context adheres to the mathematical apparatus defined in [Analyti *et al.*, 2007], however here a context is a set of concepts not objects.

Definition 2 (Context)**.** A context consists of a context identifier and a set of concepts identifiers.

Based on the mechanism for identifying concepts context is identified respectively by recursively identifying all the constituent concepts.

The notion of context supports a series of features as they have been defined in [Analyti *et al.*, 2007]:

- **concept sharing or overlapping contexts**: a concept can belong to one or more different contexts;
- **context-dependent concept names**: same concept can have different names based on the context;
- **context dependent references**: same concept can have different references within different contexts;
- **context sharing**: two different concepts can have the same reference no matter whether they belong or not to the same context;
- **context-dependent reachability**: from within a given context, we can "reach" any concept that belongs to the reference of a concept within that context;
- **synonyms, homonyms, anonyms**: the same concept can have different names in the same context (synonyms); two different concepts can have the same name in the same context (homonyms). A concept might have no name within a context (anonyms).

Beside these features the notion of context is enhanced also with attribution, generalization and classification.

Example 3.2.2. A context in our Google calendar use case would comprise for instance the concept referring to the Google calendar service and the two concepts required to identify the Save button. Thus the system in order to be able to say that it is running in this particular context must find the three concepts that are comprised in this context.

3.2.3 Behavior

Traditionally behavior has been defined by means of *business rules* and *business processes* [Weske, 2007]. Our conceptual framework agrees with this approach. In addition we argue that behavior is strongly related to context(s) [Pascalau, 2011a].

Behavior of an entity is the set of events, actions and messages that that entity produces. An event is any observable occurrence of a phenomena. It is something that "happens", an occurrence which is detected, i.e. a `click` on a button. An event has the same meaning both in the case of a rule as well as for a process. Events can be connected to time.

Definition 3 (Rule)**.** A rule is a statement of programming logic that specifies the execution of one or more actions in the case that its conditions are satisfied [OMG, 2007].

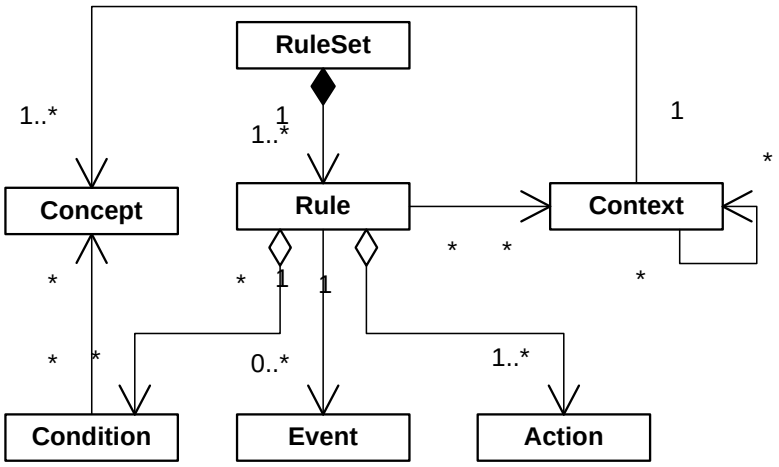


Figure 3.8: Rules

Definition 3 is also the definition of a production rule. An Event Condition Action (ECA) rule is a production rule triggered by an event. Thus the form of an ECA rule is: `on [event(s)] if [conditions] then do [action-list].`

The model depicted in Figure 3.8 is compliant with the OMG PRR specification [OMG, 2007] and is the basic model for a Rule. A Rule is related to a Context. It can be triggered by an Event, in the case of an Event Condition Action (ECA) Rule. It can be conditioned by a set of conditions. Conditions concern Concepts. Actions are the result of rule execution, basically are calls to functions.

A example of a rule in natural language is depicted in Example 3.2.3.

Example 3.2.3 (Rule Example)**.**

```
If a click event has been raised from the search button residing in the calendconf service
context and there are is a search field in this same context and the value of the search
field is found in the subject column of the DbWorld service context and there is for
each event a start date in the same row as the subject column and a location then save
this event entry.
```

There are different definitions for the notion of a process. However all agree that a process is about behavior. Weske argues in [Weske, 2007] that a *"business process consists of a set of activities that are performed in coordination in an organizational and technical environment"*, Ackoff on the other hand defines a process as a sequence of behavior that constitutes a system and has a goal producing function [Ackoff, 1971]. Our definition of a business process is enunciated in Definition 4.

Definition 4 (Business Process)**.** A business process is an ordered and coordinated set of events and activities required to achieve a user goal.

According to the BPMN specification [OMG, 2009] an activity is a general term for a work unit in a process. An activity could be atomic or non-atomic (compound).

The Process Concept is expanded in Figure 3.2.3. The definition is based on the BPMN 2.0 specification. Thus a process is a `FlowContainer` and contains `FlowElements` and `SequenceFlows`. Furthermore a `FlowElement` is either an `Activity`, a `Gateway` or an `Event`. An `Activity` is subclassed by a `SubProcess`, meaning that a `Process` might have subprocesses, and by a `Task`. A `Task` is an atomic `Activity`. However this model introduces the following relationships, which were not previously contained by the BPMN 2.0 specification: the execution of a Task can mean

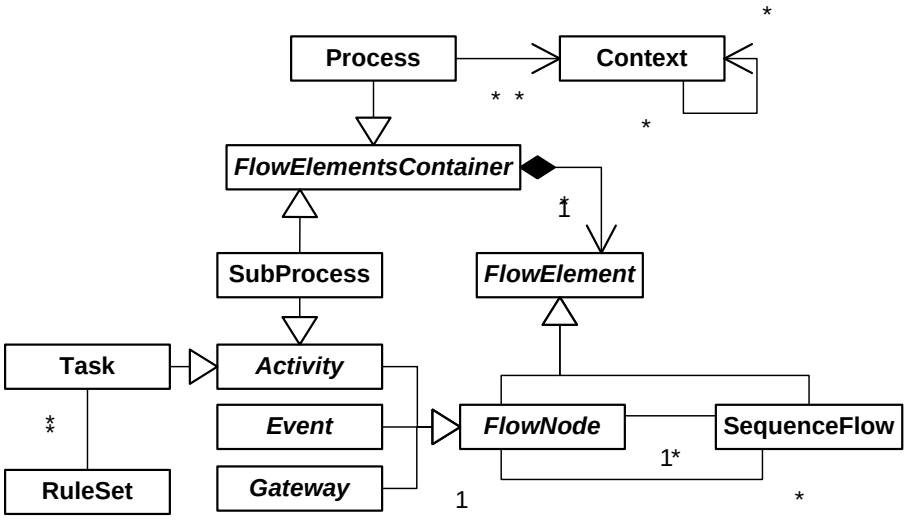


Figure 3.9: Business Process, based on BPMN 2.0 specification [OMG, 2009]

the execution of RuleSets; `Processes` are related to `Contexts`. This particular relationship can provide as discussed in [Pascalau et Rath, 2010] meaning to processes.

A rule action is the same with an atomic process activity. The difference between processes and rules is that processes are ordered while rules are not. Hence processes are executed according to the order in which constituent elements have been defined. Rules on the other hand are executed in no particular order. Therefore by means of processes, mostly execution of rule actions can be ordered.

Therefore:

Definition 5 (Behavior)**.** Behavior is a set of rules and / or a set of processes, or a combination of the two, related to context(s).

3.2.4 Mashup

This section unifies and puts together previously discussed aspects in order to define the *mashup* concept. In consequence a mashup is a plan (map) which describes the context(s) and related behavior that a user needs to do in order to achieve a desired goal. Such a mashup is defined from a user perspective. This plan created by the human user represents a common understanding of what needs to be done. This plan contains context(s) and behavior which is strongly related to the context(s) [Pascalau, 2011a]. As we discussed in [Pascalau et Rath, 2010] context provides meaning to processes. For example one could deal with a sel/buy process, a very generic one. But whenever contextual information is added, the *meaning* of a process could be totally different, as there is a big difference between selling tomatoes and selling e.g. chemical products. To support even more this idea SBVR specification [OMG, 2008] states that a body of shared meaning that a community has is represented in concepts, fact types (relationships between concepts) and business rules (constraints on concepts and fact types).

Definition 6 (Mashup)**.** A mashup is a set of contexts and behavior.

Figure 3.10 depicts the general framework. Hence the `Mashup` concept contains 1 or more `Contexts`. Further a `Mashup` can contain `Processes`, `Rules` or a combination of those two. A

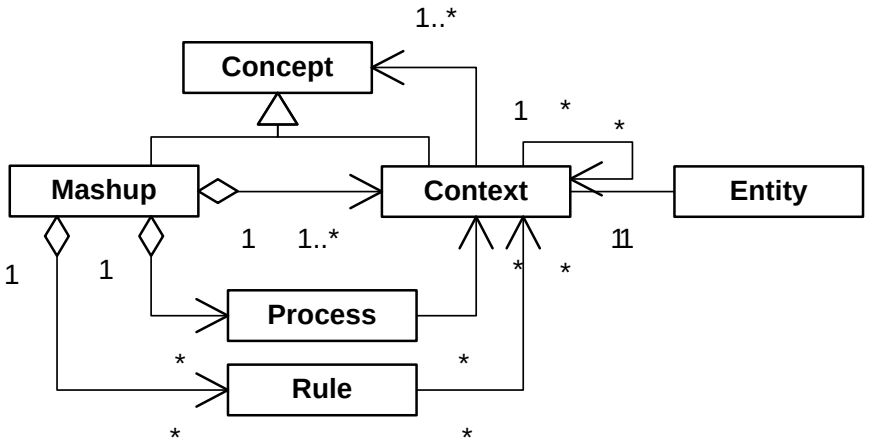


Figure 3.10: Mashup Concept

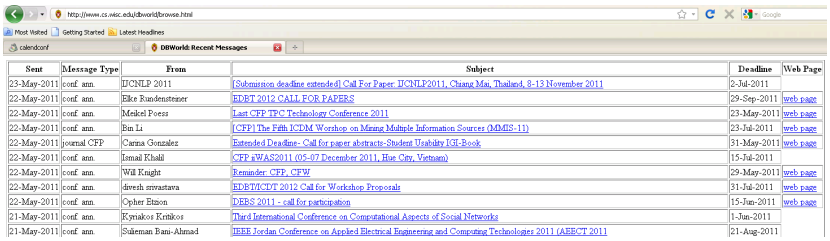
`Context` is basically a collection of `Concepts`. In addition a `Context` could have subcontexts. A `Context` refers to an `Entity`.

We argue that our model complies in general lines with (means that our approach can addresses the use cases that where addressed using the common model) the common model for mashups we extracted (see Figure 2.18 in Section 2.6).

To exemplify the use of the conceptual model, we use our recurrent conferences calendar example. Figures 3.11 - DbWorld, 3.12 - Google calendar create event facet, 3.13 - Google calendar save event facet, and respectively 3.14 - Calendconf, depict the services we need to achieve the end-user goal of storing DbWorld events in a Google calendar. Three services are involved in this use case: DbWorld, Google calendar and Calendconf. Calendconf is a user defined one. It is just a basic HTML page, who has no concrete functionality attached to it. Please note that for us any web page is also a service [Pascalau, 2011b]. Because DbWorld does not provide a search field we need a service (Calendconf) to provide this capability: a search field and a search button to start the search.

According to Figure 3.6 a service is an Entity. We have already explained that in the case of the service concept the characteristic of interest is the `URL`. All these services are therefore identified using their URLs. Next following the model depicted in Figure 3.10 each `Entity` has an association with a `Context`. Therefore for each service we have a `Context` associated with it. Contexts are set of concepts. Contexts are identified if the set of concepts defined have been identified. For example the DbWorld context is identified if we can find rows that have a particular structure, i.e. there is a subject column, followed by a deadline column and by a web page column (Figure 3.11). Similarly the Calenconf contexts needs to have at least a search filed and a search button in order to be identified. Since the contents of contexts might change over time, contexts' definitions in the user created plan need to contain only the minimal information necessary to identify them. For the Google calendar related contexts we need to have a create button to create an event and later to know that we are in the save event context, this context needs to have a series of input fields and a save button.

The rules that make the behavior of this user defined plan has been enunciated in Example 3.2.3 and states the followings: "If a click event has been raised from the search button residing in the calendconf service context and there are is a search field in this same context and the value of the search field is found in the subject column of the DbWorld service context and there is for each event a start date in the same row as the subject column and a location then save this event entry in the Calendocs context



Sent	Message Type	From	Subject	Deadline	Web Page
23-May-2011	conf. ann.	IJCNLP 2011	[Submission deadline extended] Call For Paper: IJCNLP2011, Chiang Mai, Thailand, 8-13 November 2011	2-Jul-2011	
22-May-2011	conf. ann.	Elke Rundensteiner	EDBT 2012 CALL FOR PAPERS	29-Sep-2011	web page
22-May-2011	conf. ann.	Meikel Poess	Last CFP TPC Technology Conference 2011	23-May-2011	web page
22-May-2011	conf. ann.	Bin Li	[CFP] The Fifth ICDM Worshop on Mining Multiple Information Sources (MMIS-11)	23-Jul-2011	web page
22-May-2011	journal CFP	Carina Gonzalez	Extended Deadline- Call for paper abstracts-Student Usability IGI-Book	31-May-2011	web page
22-May-2011	conf. ann.	Ismail Khalil	CFP iWAS2011 05-07 December 2011, Hue City, Vietnam)	15-Jul-2011	
22-May-2011	conf. ann.	Wil Knight	Reminder: CFP, CFW	29-May-2011	web page
22-May-2011	conf. ann.	divesh srivastava	EDBT/ICDT 2012 Call for Workshop Proposals	31-Jul-2011	web page
22-May-2011	conf. ann.	Opher Etzion	DEBS 2011 - call for participation	15-Jun-2011	web page
21-May-2011	conf. ann.	Kyriakos Kritikos	Third International Conference on Computational Aspects of Social Networks	1-Jun-2011	
21-May-2011	conf. ann.	Suleiman Bani-Ahmad	IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies 2011 (AEECT 2011	21-Aug-2011	

Figure 3.11: DbWorld Browse

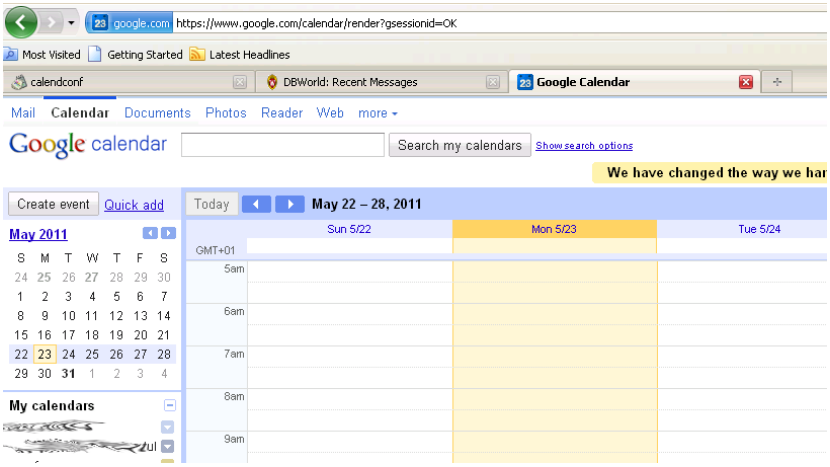


Figure 3.12: Google Calendar - create event facet

for review." After the human user reviews the results of the search and selects the desired events then a second rule is run when the save button in the Calendconf context has been clicked. Saving an event in the Google calendar requires a click on the create event that forwards the end user to the save event context, where all information concerning an event can be entered. Secondly, this information needs to be inserted in the correct fields and then the event can be saved. Hence this can be achieved with two rules. One rule will take each selected event from the reviewed list of events and the second one will be triggered as a result of executing the first rule. These rules are constructed similar to the one in Example 3.2.3.

Therefore the user has created the plan that contains a description of all the services involved, the contexts and the set of rules that are related to these contexts. Now this plan is followed by both the human user as well as by the intelligent system. The system reacts to the actions that the human user performs in the environment. And the human user at its own turn reacts to the changes that the system performs.

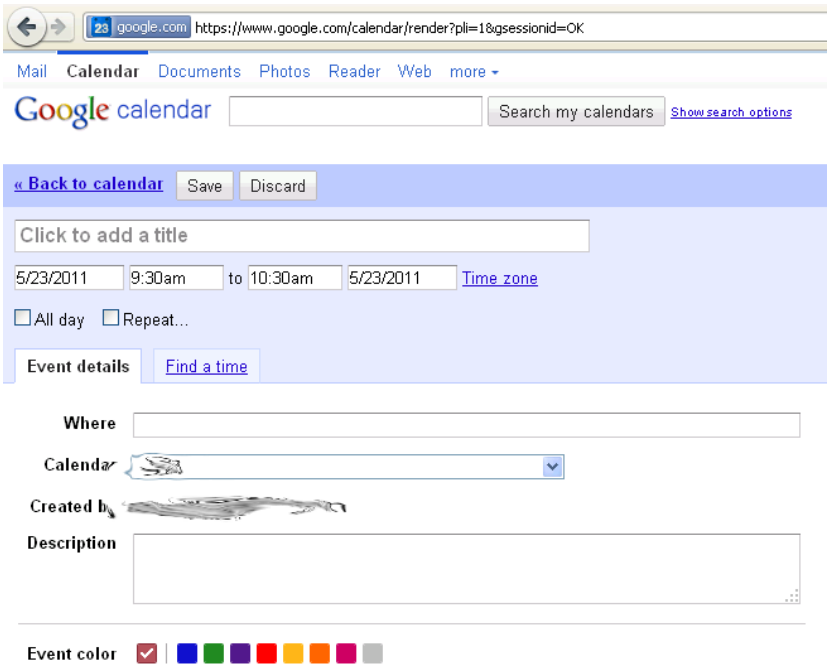


Figure 3.13: Google Calendar - save event facet



Figure 3.14: Calendconf Service

Chapter 4

Architecture

The architecture we propose in this chapter follows the approach we have just introduced previously. To the best of our knowledge there exists no system related to the Web context: (1) that follows a two layer approach to system design; (2) that uses a single representation that is understood both by the user and the system in the same way; (3) that considers and requires the end-user to actively participate in the system in order to achieve a desired goal.

In spite of this void in the Web related context - which we argue that can be filled with an approach similar to the one introduced in this thesis - there exists an approach similar in characteristics in a different field. The TomTom[1] system is probably one of the most known and most evolved [TomTom, 2007] Global Positioning Systems (GPS)[2]. Hence TomTom is an instance of the GPS metaphor we specified in the introductory chapter of this thesis. The architecture we propose inherits from the TomTom devices.

We are not the first ones to underline that TomTom functionality and characteristics are to be desired not only in navigation but also in other fields. For instance van der Aalst argues in [van der Aalst, 2009], [van der Aalst, 2010] that some TomTom functionalities are helpful to improve Business Process Management Systems (BPMSs). While in the case of a car navigation system such as TomTom, the driver is always in control as the GPS device does not try take control, Business Process Management Systems lack such features [van der Aalst, 2009]. Consequently the navigation system of a car provides a continues overview of the current situation (i.e. traffic jams, speed) but this type of information is typically missing in BPMSs.

The rest of this chapter is organized as follows: first we are going to discuss side by side our approach which we introduced in Chapter 3 and the GPS metaphor we introduced in Chapter 1. We will explain why the GPS metaphor is similar to our approach. Then we will discuss how the TomTom (GPS) system works. Afterwards we will present our architecture and discuss other aspects that are inherited from cognitive systems, especially from the Human Processor defined in [Newell, 1994], from multi agent systems (MAS), Service Oriented Architectures (SOA) and Distributed Oriented Architectures (DOA). We will address as well why a web browser based solution.

4.1 Our Approach vs. GPS metaphor

Figure 4.1 depicts the approach we introduced in Chapter 3. Hence what our approach proposes is a system that combines the human user as well as an intelligent system, who need to work together in

[1] http://www.tomtom.com/

[2] http://en.wikipedia.org/wiki/GPS

order to achieve the desired goal of the human user. The goal is not explicitly defined, but the goal fulfillment means the execution of a plan. The Human user provides the intelligent system with a plan that is understood in the same way both by the human user and the intelligent system. However the human user who created the plan does not necessarily need to be the same human user who is using the system. The plan is a description of the environment (contexts), as well as behavior associated with these contexts. The behavior defined in the plan, describes how the system needs to act and react according to changes that appear in the environment. The human user as well, follows the same behavior described in the plan. The intelligent system reacts only if the defined behavior and context(s) description is present.

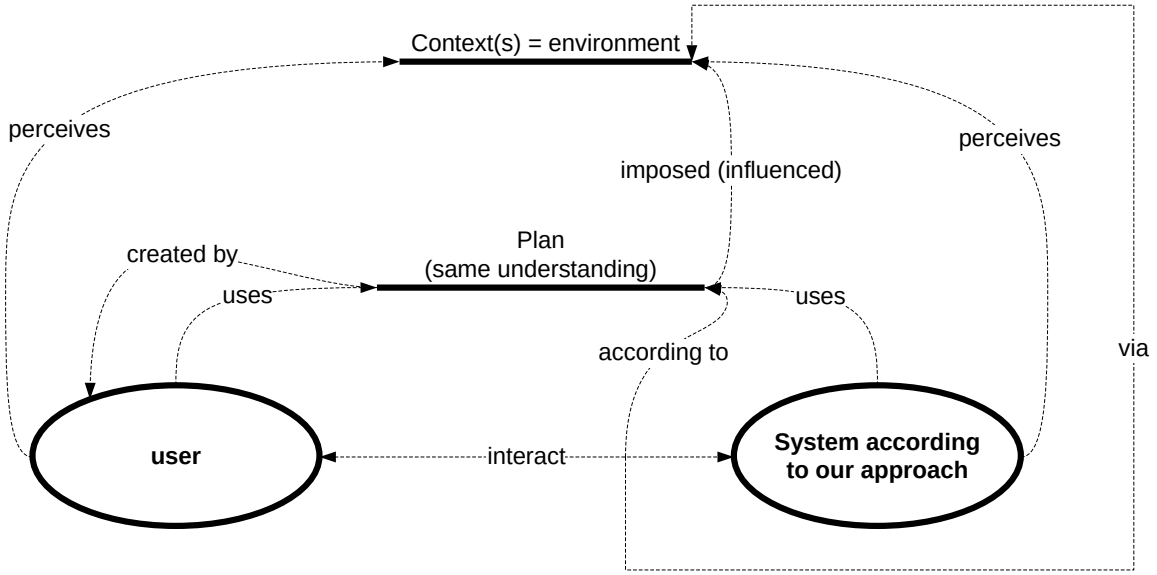


Figure 4.1: Our approach

On the other hand, in Figure 4.2, is depicted the GPS metaphor of a complete system required to get a person, for example, from Paris to Berlin. The actors involved are the human user (who is driving a car) and the intelligent system which in this case is the GPS system. Both the human user and the GPS use a plan (here a map) to achieve the desired goal of getting to Berlin. The map contains the description of context(s) i.e. cities, roads, etc. Behavior is also specified. A particularity of this system is that in order to define behavior, the user either directly inputs this behavior by specifying exactly which route to follow and in this case the GPS system will verify if the human user has deviated from the defined behavior. Or, the second possibility is just to define the start and end location and the GPS will compute the route that needs to be followed by the human user. The end-user is also part of the system, because if the user does not drive the car, the desired goal will never be achieved. At very moment the GPS system perceives the environment by receiving events from the satellites and thus being able to continuously computing current position.

We believe that similarities between our approach and the GPS metaphor can be easily observed. The GPS metaphor follows as well the two layer system we identified for our approach. Similarly a plan, which has been defined by the end-user and then given to the intelligent system, is used. The plan (the map) in the GPS case comprises also a definition of the context(s) and the behavior that needs to be followed both the human user and the intelligent system. The plan is understood in same way by

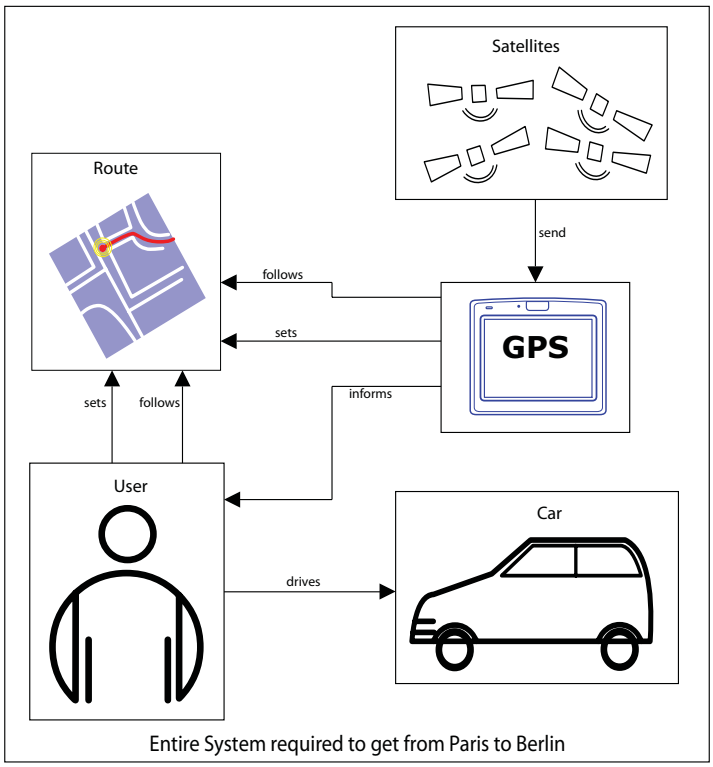


Figure 4.2: GPS Example

all the parties involved. Behavior is perceived also through changes that appear in the environment (contexts). Both the human user and the system are required, as active components of general system, in order to achieve end-user's desired goal. Therefore we argue that the GPS metaphor is a precise instance of our approach.

Next section will present the components of the GPS system. We will use this in the design of the architecture we propose.

4.2 TomTom

The GPS resolves a simple problem (intelligent navigation) compared to the Web. Navigation is an ancient issue and before the technological advance people where navigating by reading the position of stars, land based signs and paper based maps. Moreover navigators were required to know the algorithms necessary to compute position based on natural elements i.e. stars. Nowadays all these low level technical issues are hidden from the end user, exactly as per our requirements. Today only, digital maps are presented to the user. Maps and all the elements defined and depicted on them, as well as behavior are understood and shared between the end user and the GPS.

4.2.1 Digital maps - The plan

A map is a visual representation of an area - a symbolic depiction highlighting relationships between elements of that space such as objects, regions, and themes[3].

[3]http://en.wikipedia.org/wiki/Maps

Digital maps can provide users with way much more information than any traditional map. Traffic signs, prohibited manoeuvres, vehicle restrictions, post/zip codes, house number ranges, points of interest, tourist information etc are just simple examples of what type of information can be provided by a digital map. In addition to this abundant amount of information digital maps provide also other type of services such as route calculation, route guidance.

We have already indicated that a GPS map is equivalent with the plan for our approach.

4.2.2 How the GPS System Works

The GPS system as whole is the the first satellite navigation system, developed by the U.S. Government's Department of Defense and was lunched in 1973. Instead of using stars as reference points the GPS uses a constellation of artificial satellites.

TomTom system (GPS receiver) comprises mainly a GPS module and digital maps. Beside these it provides also a user interface (through which a user can define his/her destination points, etc.), as well as other artifacts. The GPS and the digital maps interact with each other in the sense that position computed by the GPS is displayed on maps.

Basically the GPS receiver perceives signals sent from 4 satellites and computes its location, which is relative to the position of these 4 satellites. Satellites orbit outside of the earth atmosphere on fixed paths. Location is computed based on synchronized timing between the satellites and the GPS receiver as well as on long-term general information about the position of the satellites constellation satellites. It is also important to mention that the satellites transmit continually signals and GPS units receive them continuously.

4.2.3 Outcomes

Therefore we argue that our approach is 100% compliant with the TomTom system and vice-verse. Moreover we learn that the TomTom system is heavily event based.

From our point of view the most important aspects that we learn from the TomTom device are: first that, the low level layer of the system (system level) has been designed by default as real time system and second that at the system level there is a unified way of representing information. We argue that these two aspects provide the basement and are fundamental requirements for building systems that comply with the approach we introduced. Being real time allows behavior specification. The unified way for information representation is the requirement for the end-user defined plan and for building an intelligent system than can be aware of itself and of the environment.

4.3 The Architecture

The architecture we introduce here concerns web browsers, because of several reasons. First of all, in accordance with the last comments that ended the preceding section web browsers are indeed real time, and they also offer a unified way for representing information. In addition to the unified of representing information web browsers provide also a unified way to program and access it. Furthermore end-users almost all of them know how to use a web browser; a web browser exists almost on all actual devices, ranging from game consoles, TV receivers, smart phones, tablets, desktop computers. The technology is the same everywhere; entire operating systems have been build as a web browser, see for instance Google Chrome OS[4], Firefox OS[5]; a large number of SDKs for building cross-platform mobile apps

[4] http://www.chromium.org/chromium-os

[5] http://www.mozilla.org/en-US/firefox/os/

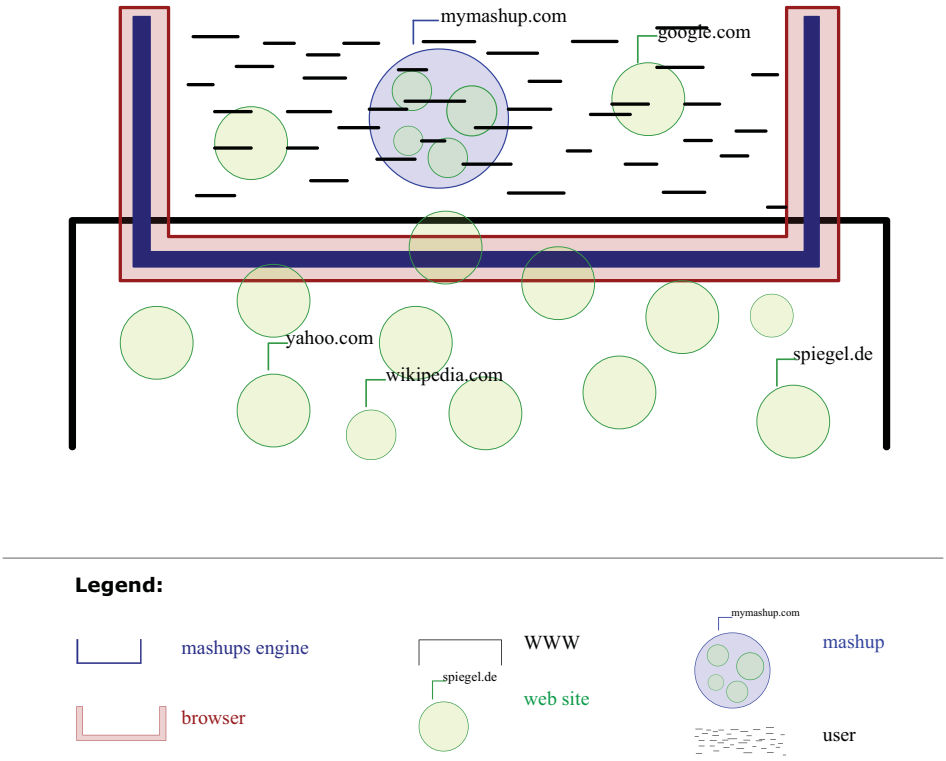


Figure 4.3: The General Architecture

are build using HTML5+JavaScript. See for instance Sencha Touch 2[6], PhoneGap[7], jQuery Mobile[8].

The general architecture of a system that complies with the approach we introduced (see Figure 4.1) and which is similar also with TomTom devices is depicted in Figure 4.3.

This architecture is tailored for the web. As it can be seen from the image the mashup engine is part of the browser, thus giving it elevated access to the content both of the browser itself and to the web pages that can be accessed via the browser.

One can picture the whole system as a special type of *aquarium* that has no bottom. The browser enriched with such a mashup engine it is sunken into the water. Thus the user of the browser and consequently the user of the mashup engine have access to all the services that the browser and mashup engine have access. Both the user of the browser as well as the mashup engine "look" at services as out of a box. From inside the box one cannot see too much. But from outside the box one can see everything is inside the box as well the box itself. Here the same principle is applied.

Figure 4.4 depicts a more detailed view of the position that the mashup engine has in relation to the web pages (and any other service) that the browser accesses. In principle web pages accessed via a browser have several layers. There is the JavaScript and Ajax layer. As depicted in Figure 4.4 JavaScript code can be splitted into several files and loaded from several locations. The same rule applies to Ajax objects. Then there is the Document Object Model (DOM) [Hors *et al.*, 2004] layer. No matter what representation is at the top, inside the browser that representation is a DOM. This is the unified representation of information that we emphasized as a fundamental characteristic to be able

[6]http://www.sencha.com/products/touch/
[7]http://phonegap.com/
[8]http://jquerymobile.com/

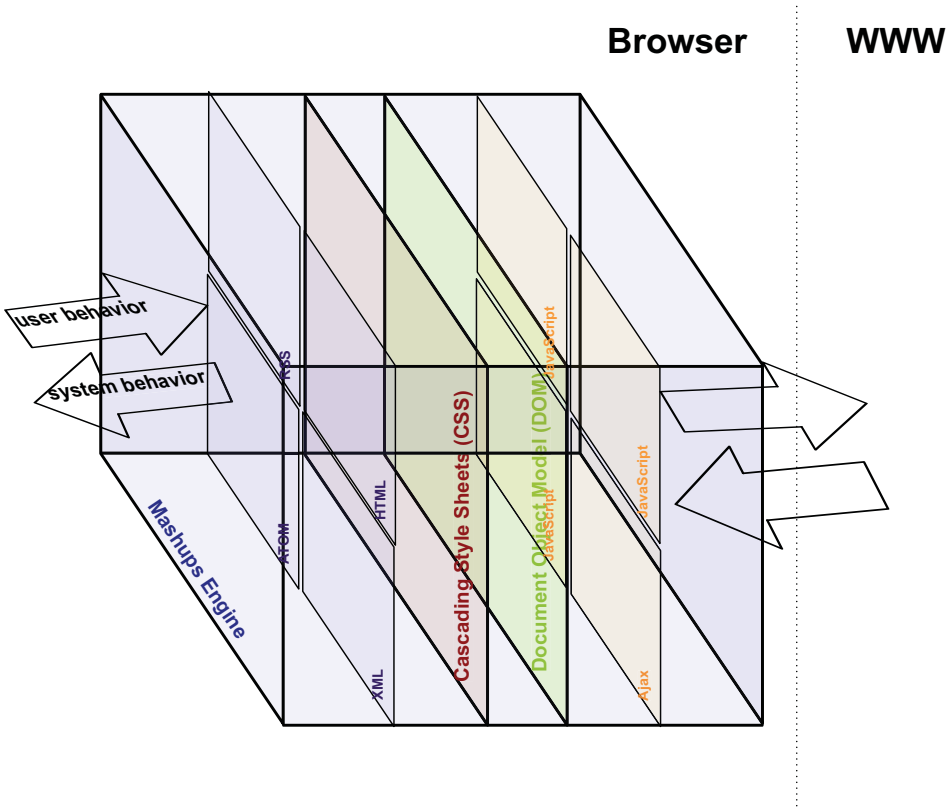


Figure 4.4: Browser Layers

to implement a system compliant with our approach. In some cases (i.e. Firefox[9] uses XUL [10]) and hence even the browser itself is a huge DOM. An important aspect about the DOM is that is completely event based. Any interaction, any change is signaled via a DOM Event [Pixley, 2000]. In consequence we have also the second fundamental requirement, as we identified while investigating the TomTom device. A third layer is the Cascading Style Sheet (CSS) [Bos *et al.*, 2010] layer. This layer applies styling. Forth layer is the representation: either a basic web page, an XML document, ATOM etc. The mashup engine as depicted is a box that comprises all these layers. Thus it has access to all of them. Any interaction, conversation, collaboration that exists between the user, browser and any of the services creating a mashup is done via actions and events. The set of actions and events form the behavior. The user is represented in the system via its behavior (actions and events).

The mashup engine depicted in Figure 4.5 has a set of sensors and a set of actuators. Via the sensors the engine perceives any interactions and communications with the environment. Different sensors and different actuators can be implemented. The default sensor `DOMSensor` listens for DOM Events. The engine uses the actuators to modify and / or communicate back with the environment. Default actuator `DOMActuator` performs actions that concern the DOM, i.e. insert text into a text field in a form. The environment comprises an arbitrary number of services and mashups, the browser and the user (user is represented via its interaction with the system), and has the following characteristics:

- *accessible*: the entire environment is accessible to our intelligent system, because sensors have

[9] *http://www.mozilla.com/en-US/firefox/*
[10] *http://developer.mozilla.org/En/XUL*

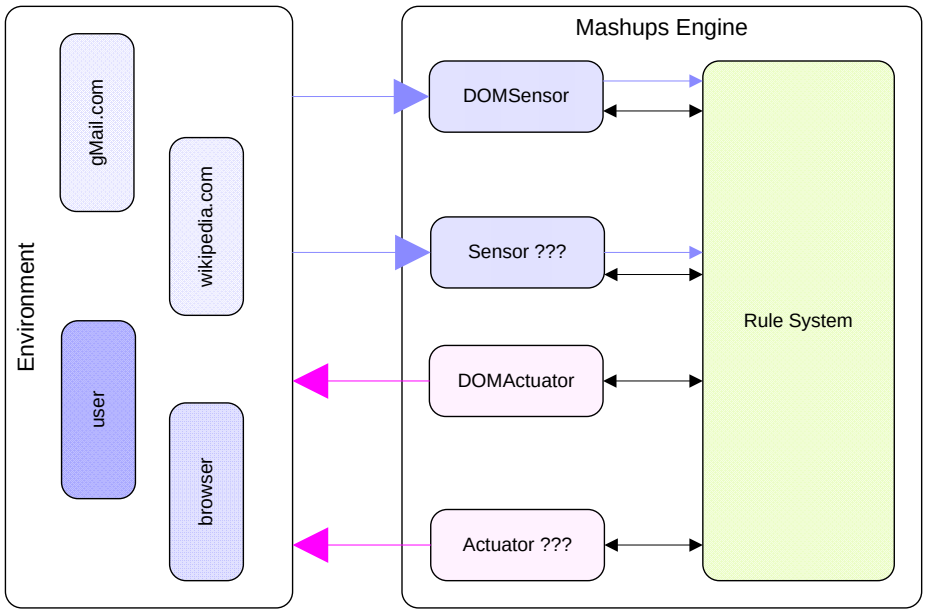


Figure 4.5: Mashups Engine

access to all required information to be able to chose an action [Russell et Norvig, 2009];

- *deterministic/nondeterministic*: depending on the reasoning mechanism and because of the high degree of complexity the environment can be seen both as deterministic and nondeterministic from the point of view of the system (agent); however the default approach will be deterministic;
- *episodic / nonepisodic*: the environment can be both episodic and nonepisodic. Episodic after [Russell et Norvig, 2009] means that an agent perceives and then acts, and subsequent episodes do not depend on what actions occur in previous episodes;
- *dynamic*: the environment is dynamic because it can change while the agent is deliberating;
- *Discrete / continuous*: the environment can be both discrete and continuous depending on the use case and on the *plan* provided.

All the events perceived, are processed by the *Rule System*. Based on the knowledge available, the mashup definition and the events perceived, the Rule System computes the set of actions that the engine has to perform in the environment. The Rule System receives events that come from the environment as well as events that come from the rule system's internal components. Both external and internal behavior is performed via events and actions. The overall architecture is the one of a real-time system as discussed in [Sommerville, 2007], Chapter 15.

The Rule System of the mashup engine comprises two event processors one that deals with external events and the other one that deals with the internal events. *WorkingMemory* stores knowledge over time. The interaction between the components is mediated by the *WorkingMemory*. The *Inference Engine* is the component in charge with the reasoning processes (basic functionality is pattern matching). Different reasoning algorithms can be plugged-in.

Our architecture concerns an intelligent system and therefore the mashup engine has influences from multi agents systems [Russell et Norvig, 2009] and from cognitive sciences [Newell, 1994] -

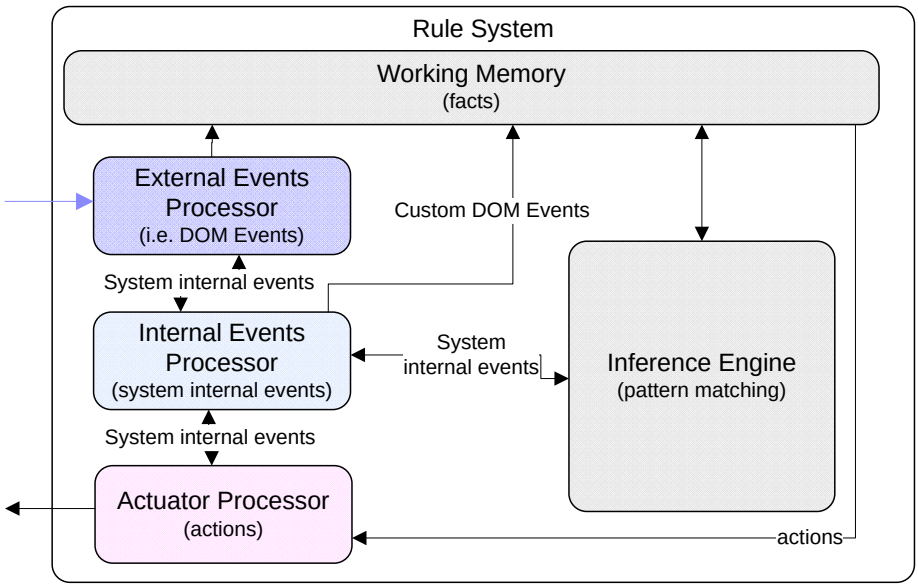


Figure 4.6: Mashups Engine - The Rule System

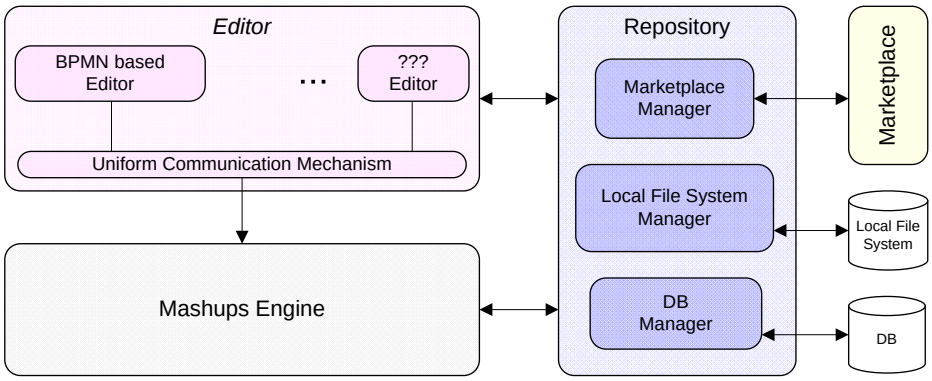


Figure 4.7: Framework Architecture

especially the Model Human Processor (MHP) block diagram introduced in [Newell, 1994]. The behavior that the introduced architecture is capable to tackle, is as general as the W3C standards define. Thus all DOM Events and actions defined by the [Pixley, 2000] are taken into account. The *DOMSensor* and *DOMActuator* are responsible for these. An arbitrarily number of sensors and actuators can be defined. In addition, for example, browser specific sensors can be defined. Sensors and Actuators can be defined also for frameworks such as YUI[11].

Figure 4.7 depicts the architecture of the entire framework. It comprises three major components: the mashup editor, the mashup engine and the repository. The mashup editor it is foreseen to be also browser based. The framework does not even constrict the editing environment to a particular language or editing approach. As it can be seen in Figure 4.7 the editor must implement a *Uniform Communication Mechanism*. This mechanism acts as a translator between what ever language or approach

[11] http://developer.yahoo.com/yui/

are used by the editor and the mashup engine which knows its executable language. The conceptual model of our solution has as smallest unit of knowledge the concept notion (see Section 3.2.1). This allows great freedom to experiment and to address different type of users when it comes to visual representation and modelization of mashups. For example for those who have a strong background in business then a business processes, choreographies and business rules would be most suited while for the segment of users that have a background in cognitive sciences and psychology perhaps a editor capable of dealing with mental representations would be better. The second component is the mashup engine which we just discussed. The repository, which sores and loads the masups definitions or the *"user plans"* comes in different flavors: data base (DB), local file system and market place. These different types address different types of users: private users, and business users. Business users can use either the market place or their own DB if the business user is an enterprise wishing to provide own employees with a uniform set of applications. The context/ontology representation approach is also beneficial for repositories, allowing for context based and meta search facilities [Pascalau et Giurca, 2009a].

4.4 Discussion

4.4.1 Web pages as Web services

We argue that any web page should be considered as a web service and used accordingly. And the web browser based mashups solutions supports this idea. This vision is also shared by others. However there web pages are transformed before they can be mashed up[12]. Another example discussed in [Ennals *et al.*, 2007] uses user defined extractors for web pages. These extractors are used by the mashup tool to get access to the content. On the other hand, the solution we propose in this thesis uses web pages natively without any other transformation into intermediary formats, or without requiring any sort of annotations.

Sommerville [Sommerville, 2006] defines a Web service as a loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols.

A more general definition given by Lovelock [C. Lovelock, 1996] states that a service is an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production.

According to this last definition any web page can be considered as a Web Service. Moreover we believe that a web page complies also with the Web Service definition given in [Sommerville, 2006] as a loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed.

Hence our Web browser solution is concerned with all web services that can be accessed through a web browser. In addition because some browsers such as Mozilla Firefox [13] use the same information representation for themselves (uses XUL [14] which is DOM based), our solution targets also the browser, meaning that end-users could for instance personalize the functionality of the browser.

Being able to use also any web page as a web service our approach complies with requirement R6.

[12] http://www.dapper.net
[13] http://www.mozilla.com/en-US/firefox/
[14] http://developer.mozilla.org/En/XUL

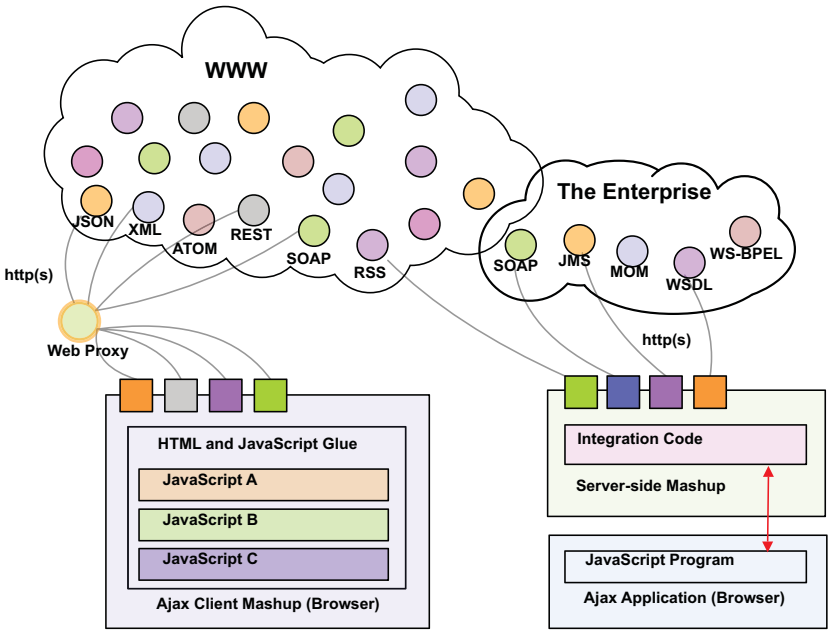


Figure 4.8: Mashups styles. Inspired from [Valica, 2007]

4.4.2 Mashups styles

Figure 4.8 recapitulates briefly the mashups composition styles that have been discussed at large in Chapter 2.

Typically, mashed-up applications are created by means of lightweight components - known as widgets, or gadgets. These components have to be previously created, in order to be used. Wicks et al. characterize a widget as "a small program or piece of dynamic content that can be easily placed into a Web site. "Mashable" widgets pass events, so that they can be wired together to create something new", [Wicks *et al.*, 2009]. Typically widgets are defined declaratively using an XML based language, but the generated executable code is platform dependent. A major downside of this approach is that in order to have two or more widgets exchanging information all of them must run on the same platform. This is required because of security reasons and because the widgets communicate via a container. The widget technology it is said to be a client based approach. However we consider that is in fact a hybrid approach: a client-server approach, because the gadgets run on a server side platform, and all the events and data that is exchanged between the gadgets is performed via a server side mechanism. Nevertheless from a compositional perspectives, users combine them in a client (browser) based environment.

Another client side approach is the API based. As depicted in Figure 4.8 JavaScript APIs are used together to glue either data, which comes in different formats (i.e. ATOM, JSON), or presentation. An important aspect to emphasize is that all APIs go through a web proxy, for security reasons. Thus again, from the author's perspective we are dealing with a hybrid approach as another server is involved. Indeed the composition is done on the client side via JavaScript but still a proxy intermediates the conversations. With respect to protocols and architectural styles the most used one is REST, but SOAP is also an option.

Compared to this composition styles we argue that our proposal is pure client based mashup solution. Moreover via the arcthitecture we proposed the end-user is not required to install and manage any heavy solutions such as servers; end users are not required to run code on proprietary platforms

nor are they required to run code through proxy servers in order to get access to content. We believe that as required by requirement R6 this architecture provides the means to have the web as an open application execution environment.

4.4.3 DOA and SOA influences

Service Oriented Architecture(SOA) by OASIS:

> A service is a set of functionality provided by one entity for the use of others. It is invoked through a software interface but with no constraints on how the functionality is implemented by the providing entity... A service is opaque in that its implementation is hidden from the service consumer except for (1) the data model exposed through the published service interface and (2) any information included as metadata to describe aspects of the service which are needed by service consumers to determine whether a given service is appropriate for the consumer's needs

Distributed Objects Architecture(DOA) by OMG:

> Distributed object applications are composed of objects, individual units of running software that combine functionality and data, and that frequently (but not always) represent something in the real world. Typically, there are many instances of an object of a single type... For each object type you define an interface. The interface is the syntax part of the contract that the server object offers to the clients that invoke it. Any client that wants to invoke an operation on the object must use this interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the same definition is used there to unmarshal the arguments so that the object can perform the requested operation with them

The definitions themselves share many similarities. Both approaches are structured around remote entities, either services or objects. In both cases remote entities export typed interfaces. Invocation by clients at remote sites is provided by both approaches. As concluded in [Baker et Dobson, 2005] the point of both architectures it to provide *interoperability* rather than *homogeneity*.

DOA provides a stable computing platform for the enterprises. However to extend DOA for business-to-business interactions proved to be a very difficult task [Baker et Dobson, 2005], hence SOA has been developed both as a complement and / or based on the situation as a replacement for DOA. SOA has been introduced to be more technological neutral and to tackle the highest-level integration task. In opposition, DOA can be very specific to one specification or standard [Baker et Dobson, 2005].

[Baker et Dobson, 2005] compares DOA and SOA architectures in detail. The outcome states that there are both differences and similarities between distributed object architectures and service oriented architectures. Four major differences have been identified: (1) SOA supports a more coarse grained aggregated interfaces in order to simplify interactions across enterprises. Thus a smaller number if interfaces has to be understood and this is an advantage in a world where businesses interact with increased dynamics; (2) object references can be exchanged within the framework; as such different providers can work with each others data; (3) reducing the importance of interface types does not lead on the long term to simpler integration; (4) business integration has not started with SOA, however SOA provides a better understanding for industries and commonalities that exists between providers.

Web services are distributed technologies, but are about interoperable document centric computing, not distributed objects [Vogels, 2003].

In the browser DOA interfaces are the JavaScript APIs. Through these APIs applications have access to distributed and remote objects. Same as in J2EE these APIs work similar to the exposed Java beans. Another interesting fact about the browser and here browser based mashups is that it, in itself, the browser is a singular platform providing a singular mechanism to exchange information and to access remote objects via JavaScript APIs.

Web Services are said to be able to use lightweight HTTP interactions [Baker et Dobson, 2005], but they are not tight to HTTP or WSDL. Browser-based mashups respect the requirement of *light weight programming models*. First the HTTP(s) protocol is used contrasting with CORBA's heavyweight IIOP and IDL and second browser based mashups do not require CORBA's interoperable object references (IORs).

The browser complies by default with the technological neutrality of SOA. In the browser every page (service) is presented in the same way, no matter what technology has been used on the server side (e.g. PHP, JSP, ASP etc.).

While SOA uses an explicit message based invocation mechanism [Baker et Dobson, 2005] DOA uses a "push" model, thus the receipt of an event triggers the execution of handler code. The architecture discussed here deals with first-class events which are manipulated programmatically.

In consequence this architecture is compliant with the specific SOA requirements and hence fulfills the requirement R7. Moreover as we stated in the preceding paragraphs this architecture has also DOA characteristics and therefore complies also with requirements R8. In addition because javascript is an interpreted language we argue that our architecture complies as well with requirement R9.

Chapter 5

Execution

Current chapter concerns execution of Web based applications that follow the approach and the conceptual model we introduced in Chapter 3. To recapitulate execution is based on rules and processes and is tailored for web browsers.

5.1 DOM and DOM events

"The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page." [1] Thus even if a web page is represented via HTML5, XML, XHTML or any other XML based variant, at the low level (program level) they are represented using DOM platform- and language-neutral interface.

"DOM Events on the other hand is a generic platform- and language-neutral event system which allows registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event." [2] DOM Events complement DOM. Moreover any DOM change is signalized through a DOM Event.

Therefore our execution engine uses directly these platform and language neutral interfaces. In consequence our execution engine is platform and language neutral. Moreover we argue that through the combination of a rule based execution engine and the use of these platform and language neutral interfaces can be achieved a unified and generic approach for defining and execution of new end-user defined applications that comply entirely with the list of requirements we identified in Section 2.6.

5.2 The Rule Engine

The reasoning component of a standard Production Rules System is the Inference Engine (same in our case, see Figure 5.1). The Inference Engine matches facts and data against rules to infer conclusions which result in actions. While basic production rules, consists of two main parts: (1) conditions and (2) actions, we are dealing mostly with reaction rules or ECA rules which comprise three parts: (1) event, (2) conditions and (3) actions. Both types of rules use First Order Logic for knowledge representation.

[1] http://www.w3.org/DOM/

[2] http://www.w3.org/TR/DOM-Level-3-Events/

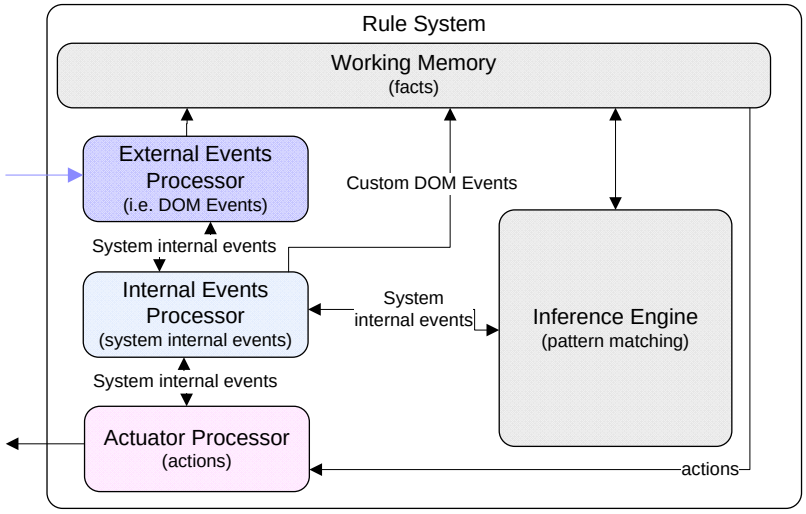


Figure 5.1: Mashups Engine - Rule System

The process of matching, new or existing facts against rules is called pattern matching and is performed by the Inference Engine. During the last thirty years there were various proposals such as RETE [Forgy, 1982], TREAT [Miranker, 1987] and the Gator algorithm [Hanson et Hasan, 1993] which is derived from the other two. However, none of them are able to directly process DOM Events and / or the Document Object Model. Moreover default implementations do not take into consideration events. Our implementation is a variant of ReteOO which addresses Object Oriented systems. More specifically closure based ones (JavaScript). In addition it takes care by default of DOM Events.

While in the case of a standard production rules system, rules are said to be stored in the production memory, in our approach rules are stored in the *user defined plan (mashup)*. In other words the production memory in our case is the user defined plan or the mashup. Facts, on the other hand, in a normal production system are asserted in the Working Memory (Figure 5.1), where they may be modified or retracted. For example if we area dealing with a database system, in order to be able to have access to the facts, all necessary facts need to be first extracted from the database, asserted into the Working Memory and only afterwards they can be used by the Inference Engine to match them against the rules.

However the big difference between standard implementations and our own implementation with respect to the Working Memory is that the `Working Memory`, for us it is the entire DOM structure (all the services, web pages that have been associated with contexts in the user defined plan) to which the Inference Engine has access. In addition because of this we do not require any assertion of facts in the Working Memory. All the facts are already in the Working Memory. In addition the Working Memory in our case holds also DOMEvents, both predefined W3C DOM events (i.e. `click`, `dblclick`, `mouseout`, `mouseover` etc) as well as custom DOM Events. All interactions take place inside the Working Memory. The Working Memory is alive as long as the mashup is active in the browser (it means that as long as in the web browser there is a tab opened in the browser that contains the mashup, the Working Memory will be kept alive).

As depicted in Figure 5.1 the Rule System has two event processors: an `External Event Processor` and an `Internal Event Processor`. The External Events Processor receives and deals with events from the Sensors (by default from the `DOMSensor`). The Internal Events

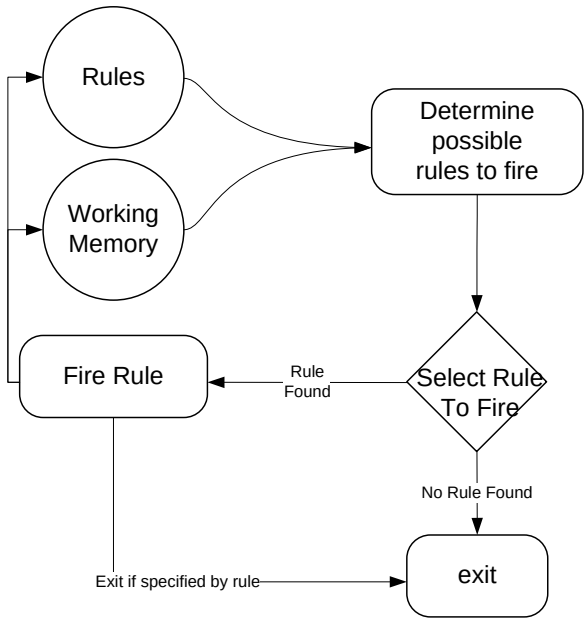


Figure 5.2: Basic execution flow of a forward chaining algorithm

Processor deals with system internal events. System components are decoupled from each other and they communicate with each other via events. The `Actuator Processor` takes care of the actions that need to be performed as result of rule matching. Actions can be any JavaScript function calls that exist in the Working Memory. Since the Working Memory contains the DOM Structure this implies also that all JavaScript objects and functions that have been defined in the web pages (services) are part of the Working Memory.

There are two execution methods for rules systems: forward chaining and backward chaining. Our implementation is a forward chaining implementation. Forward chaining is data driven, meaning that when ever there is a change in the working memory, the inference engine reacts to these changes and tries to match facts against rules. Figure 5.2 presents the basic execution flow of a forward chaining algorithm.

We are dealing with ECA rules and hence we are treating separately DOM events from the regular DOM structure. Therefore the External Events Processor as well as the Internal Event Processor use each of them, their own specific EventStack. Events are stored in the stacks in the order they are caught by the sensors (i.e. DOMSensor - in the case of predefined W3C DOM Events). The events are then processed by the Inference Engine in order to trigger rules. This is depicted in Figure 5.3.

The Rule System's main execution process is depicted in Figure 5.4. This process incorporates the basic execution flow for the forward chaining algorithm (Figure 5.2). It starts by loading the mashup. Continuously sensors listen for different types of events. Events are added to their specific stack of events by the different Event processors (external and internal). Each event is verified against all the rules defined in the mashup. If an event is found to match the event described in the rule, then the conditional part of the rule is verified against the Working Memory. If also the conditional part of the rule is found true then the action is added to the `QueueOfActions` that need to be fired. After all rules have been verified for an event then that event is deleted from the specific stack of events and the `ActuatorProcessor` will fire all the actions in the order they have been added to the `QueueOfActions`. The `QueueOfActions` is deleted after all actions have been fired.

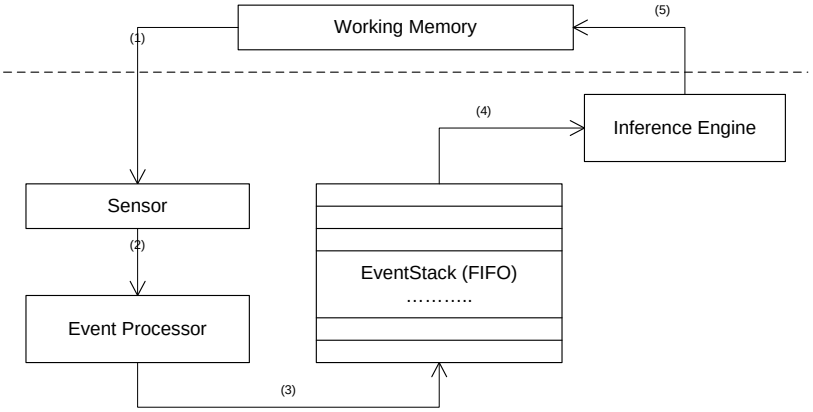


Figure 5.3: Events Capturing

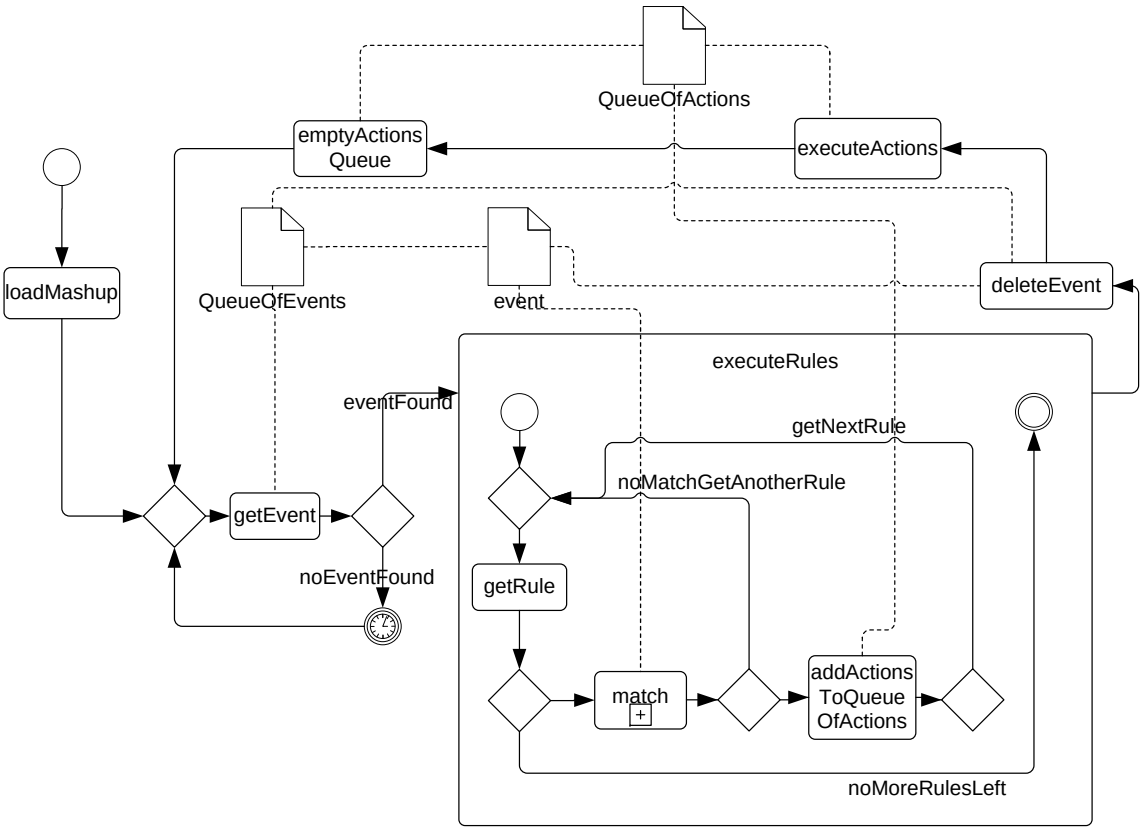


Figure 5.4: Rule System's Main Execution Process

5.3 Rules

Rules are written using First Order Logic (FOL), or predicate logic which extends propositional logic. A proposition is a statement that is either true or false. If the statement can be determined to be true or false just by itself it is said that that statement is a "closed statement": `20==4*5`.

On the other hand expressions that evaluate against one or more variables, the facts, are "open statements". Therefore the value of truth can not be determined until there exists a variable instance to evaluate against: `Event.type=='click'`.

The conclusion's action of a similar statement (`SELECT * from Events where Events.type='click'`) written in SQL is the return of the set of rows, in which for any row in the resultset we have inferred that the facts are `click` events.

For an object oriented system or closure based system such as JavaScript we say that a simple proposition is of the form *variable operator value*. Frequently we refer to a *value* as being a literal value and a proposition is a field constraint. Propositions can be combined with conjunctive (`AND`) and disjunctive (`OR`) connectives. The current implementation of our own Rule System, takes into account only conjunctive connective.

We find also important to mention that FOL extends propositional logic with new quantifier concepts, specifically universal and existential quantifiers. Universal quantifier, or forall, checks that something is true for everything in the Working Memory. Existential quantifier, on the other hand, check for the existence of something, meaning that it should occur at least once in the Working Memory.

Facts are objects (i.e. java beans, JavaScript objects). Hence for our Rule System, facts are any JavaScript objects to which the engine has access, thus any objects from the Working Memory. However only objects' fields are used in the reasoning process, or the static structure of an object: properties (field, property or attribute have the same meaning) and their values. According to the conceptual model (Figure 5.5), because all the elements of our model are `Concepts`, and in addition a concept is a subclass of `uml::Class` then any of them can be facts. In consequence rules could be used to reason about any of them in a unified way.

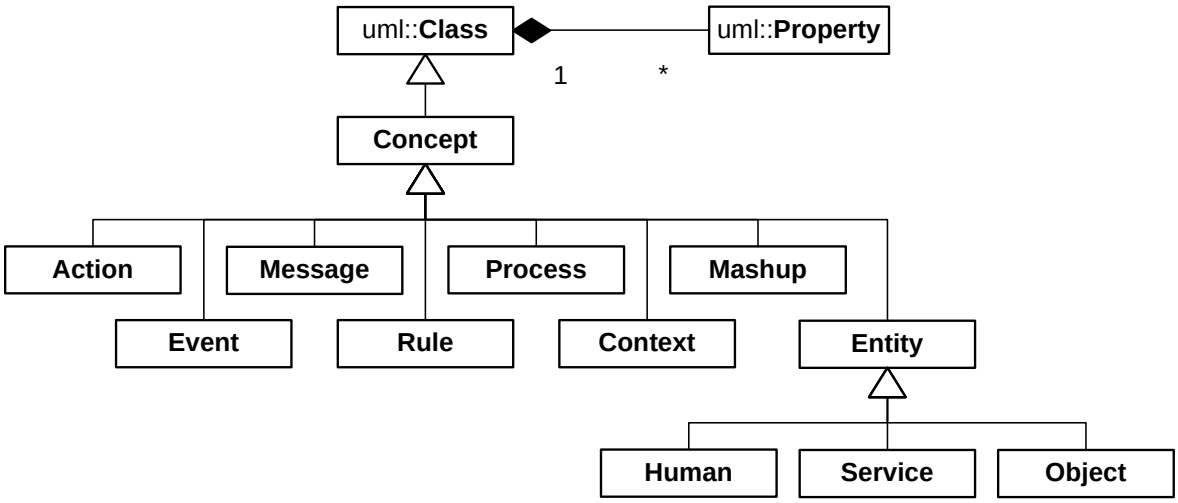


Figure 5.5: Concepts

A mashup rule is depicted in Figure 5.6 and should be read in the following way:

```
rule "id"
   refersTo (list of contexts)
   on
        event
   if
        LHS
   then
        RHS
end
```

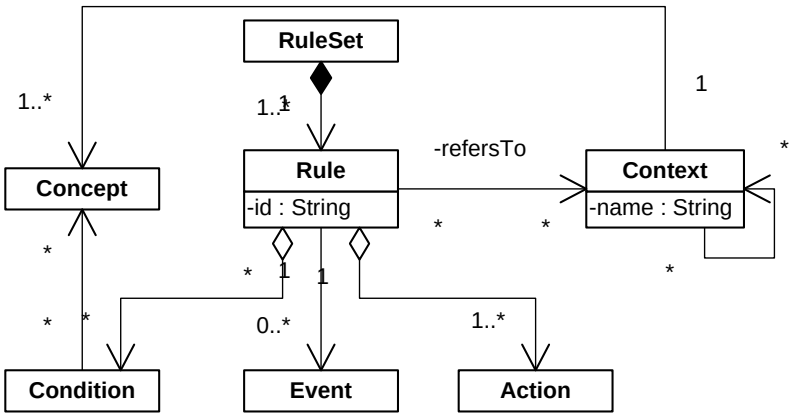


Figure 5.6: Rule

A rule specifies that `on` event caught then if a particular set of conditions occur, specified in the Left Hand Side (LHS) then do this, which is specified as a list of actions in the Right Hand Side (RHS). LHS is a common name for the conditional part of the rule. It consists of a zero or more conditional elements, as depicted also in Figure 5.6. Conditional elements refer to concepts, which in turn belong to contexts. The LHS could contain the predefined boolean value `true` which in turn implies that the LHS part of the rule is always true.

5.3.1 Event

An `Event` (see Figure 5.7) that triggers our rules is a DOM Event. Thus an event has a set of attributes as defined by the W3C standard for DOM Events: `type`, `timestamp`, `phase` and a `target`, and other attributes that are not depicted in the figure. Most used attributes are `type` and `target`. `Timestamp` as the name states refers to the time when the event occurred. The `phase` is an attribute that characterize the DOM event flow. Hence any event is propagated from the document towards the specific element in the page which was targeted by the event (from the root node towards the leaf node) [3].

The `target` of an event is the element (node) in the page on which the event has been dispatched. The `type` of an event can be any of the DOM EventTypes or any custom event that follows the DOM Event specification. Our current prototype implementation concerns only the `type` and `target` attributes. The `target` attribute could be specified or not in the event section of a rule. If no `target` is specified then it means an event of the specified type that was dispatched on any node in the page. If target attribute is specified then it can be a specific DOM node or a variable. If a variable is specified, then it is a free variable, and it needs to be specified in the LHS part of the rule. In this situation the event part of a rule will be considered true only based on the type of the event. However the LHS part of the rule will restrict the target of the event to a specific element type, as we will see later when we will discuss the different types of conditional elements that can be expressed in the LHS.

[3] http://www.w3.org/TR/DOM-Level-3-Events/#event-flow

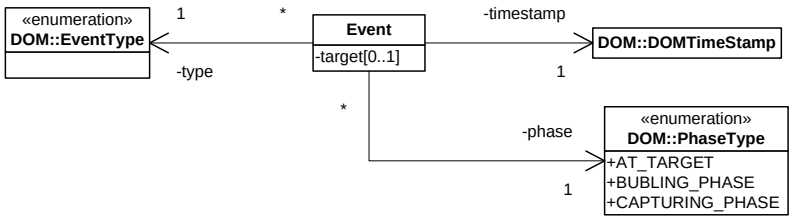


Figure 5.7: Event

Example 5.3.1 (EventDescription)**.**

```
...
"event":{
    "type":"click",
     "target":{"variable":{"name":"$X"}}
      },
...
```

A small example (in JSON format as used by the Rule System) of an event is provided in Example 5.3.1. The event described in this example is an event of type `click`. When ever a click event will be fired rules which are triggered by `click` events will be verified. The `target` of an event is the element in the page on which the event has been dispatched. Because in this case the `target` is purely bound to a free variable, named `$X`, the overall evaluation of the described event implies that any click events no matter the target (a button element, a div element or anything else) will be taken into account. However the target of the event could be restricted in the LHS part of the rule if desired so.

5.3.2 Conditional elements

The condition (LHS) part of a rule can comprise several conditional elements. We envision a series of conditional elements for our execution language: (1) a `ConceptConditional`, (2) a `JavaScriptBooleanConditional`, (3) an `EqualityConditional`. These conditional elements are already implemented in our prototype implementation. In addition (4) `XPathConditional` and (5) negation are also foreseen, but not implemented yet.

ConceptConditional

The conditional element concerning Concepts (`ConceptConditional`) is the most important one. This conditional is built to accommodate objects and the features that objects have. It actually describes how the object (concept) should look like. We identify concepts based on their properties. For example we identify our car in a parking because we know the brand (type: Renault), we know the model (Laguna), we know the color (black), we know the matriculation number (FR2014). The same process is applied here. Hence an object (concept) has a type and has properties (fields). Therefore when dealing with an object, in our case a `concept`, we need verify the `type`, we need to able to test/constrain properties' values and we need to be able to bind the value of an object property to a variable, so that this can be used later on in another conditional element of a rule. Moreover the object itself, not just a property of it, can be bind to a variable. We call constraints a `PropertyConstraint` and a `PropertyBinding`. A `ConceptConditional` can have zero or more constraints. All subelements of a `ConceptConditional` need to evaluate to true in order for the ConditionalElement to

be true.

Example 5.3.2 (ConceptConditional with PropertyRestriction)**.**

```
...
{"conceptConditional":{
"type":"HTMLButtonElement",
    "binding":{"variable":{"name":"$X"}},
      "constraints":[
        {"propertyRestriction":{"property":"id",
                                          "operator":"EQ",
                                          "value":"mashSearchButton"
                                          }
        }
      ]
    }
}
...
```

The `ConceptConditinal` depicted in Example 5.3.2 should be read and understood in this way: Forall `$X` which are of `type` `HTMLButtonElement` with the value of the `id` attribute equal to `mashSearchButton`. In order for this conditional to hold, the Rule System will search in the Working Memory all the objects that are of type `HTMLButtonElement` (all buttons from a web page) with a constraint on the id attribute. This constraint is a `PropertyConstraint`, because the id attribute of all the found HTMLButtonElements must have the value `mashSearchButton` otherwise the entire `ConceptConditional` will not be evaluated to true.

If we combine this Example 5.3.2 with Example 5.3.1 we observe that the variable `$X` has been already bound to the `target` property of the event. So when the Inference Engine will meet this variable in the `ConceptConditional` the overall meaning of the rule will be that when a `click` event is raised, then only click events that were dispatched on an `HTMLButtonElement` are of interest., all others will be ignored.

Example 5.3.3 (ConceptConditional with PropertyBinding)**.**

```
...
{"conceptConditional":{
   "type":"HTMLInputElement",
   "binding":{"variable":{"name":"$cSearch"}},
   "constraints":[
    {"propertyRestriction":{"property":"type",
                                       "operator":"EQ",
                                        "value":"text"
                                       }
    },
   {"propertyRestriction":{"property":"id",
                                       "operator":"EQ",
                                       "value":"mashSearchInput"
                                       }
    },
    {"propertyBinding":{"property":"value",
                                    "variable":{"name":"$sValue"}
                                  }
    }
   ]

                                  }
 }
...
```

A second ConceptConditional is provided in Example 5.3.3 and exemplifies a different case where a `PropertyBinding` is involved: the variable `$cSearch` is a free variable meaning not bound to any value. It will be bound to all `HTMLInputElement` instances from the Working Memory complying with the property constraints from the ConceptConditional (i.e. the `type` attribute must have the value `"text"` and the `id` attribute must have the value `"mashSearchInput"`). The `$sValue:value` a `PropertyBinding` – the variable `$sValue` will be bound to the value of the element's attribute `value`.

JavaScriptBooleanConditional

`JavaScriptBooleanConditional` is intended to be used for those cases when the other conditional elements cannot be used. This conditional is in principle a basic JavaScript boolean expression. However in our case in can contain variables. Variables need to be bound before such an expression can be evaluated. The evaluation is always a boolean value. If, for some reasons, the expression cannot be logically evaluated then the final result of the evaluation is `false`. An example (written in our rule system internal JSON format) of such conditional element looks like

```
{"javaScriptBooleanCondition":
  "new RegExp($sValue.toLowerCase()).test($Y.toLowerCase())"}
```

This example uses a JavaScript Regular Expression construct to test that the text in lower case format of the variable `$sValue` matches the text value of the variable `$Y`, also in lower case format.

EqualityConditional

This conditional element computes the equality between either two concrete objects or between two variables, e.g. `$Y == $Z`. The equality succeeds if variables are bounded to the same real object.

XPathConditional

Compared to the other conditional elements which more or less deal with constructs similar to other rule languages, this one is a bit more specific to the Web. It uses an `XPath` expression in relation with either a DOM node or a variable, to express membership i.e. an `XPathConditional` verifies the membership of a DOM node in a nodelist returned by the evaluation of the XPath expression. It is interpreted as the typical Prolog member predicate to test the membership against a list.

For example `$Y in 'child::$X'` typically checks if the value bound to $Y belongs to the result list after the evaluation of the XPath expression `'child::$X'`. Not implemented yet.

5.3.3 Actions

Mashup Rules action concept complies with the W3C RIF Production Rule Dialect (RIF-PRD), [de Sainte Marie *et al.*, 2009].

As in RIF-PRD, mashup Rules actions (see Figure 5.6) are used to add, delete and modify facts in the fact base (Working Memory). However, mashup rules allow a greater variety of actions i.e. a mashup rule action can be any JavaScript function call. By default any javascript function can have a list of parameters. We do not restrict this in any way. Moreover these parameters could be variables that have been defined in the condition part or the event part of a rule. These variables, however can not be free variables at the time of execution.

Table 5.1 maps the RIF-PRD actions to Mashups Rules actions.

Table 5.1: RIF-PRD actions mappings to Mashups Rules

RIF-PRD	**Mashups Rules**
Assert	insert a DOM Node
Retract	removes all DOM Nodes that comply with defined formula
Retract object	remove a DOM Node
Modify	update a DOM Node
Execute	any JavaScript function call

Chapter 6

Implementation

In this chapter we discuss the implementation of a system that complies with all the aspects we have introduced in the previous chapters. The prototype is our proof of concept. Moreover we will present a series of use cases in order to validate our approach. To demonstrate the expressiveness and generality of our approach, the set of use case address different topics: mashups (services); security and policies; personalization; logs.

6.1 The prototype

The prototype implementation follows entirely the architecture we discussed in Chapter 4. Hence the main components of the Rule System are those depicted in Figure 6.1. This image gets translated into the real code in a series of packages. These packages are depicted in Figures 6.2 and 6.3.

Figure 6.2 depicts the main packages: `rulesystem`, `repository`, `lang`, `utils`, `io`. The `rulesystem` package as the name states stands for the Rule System. `Repository` package

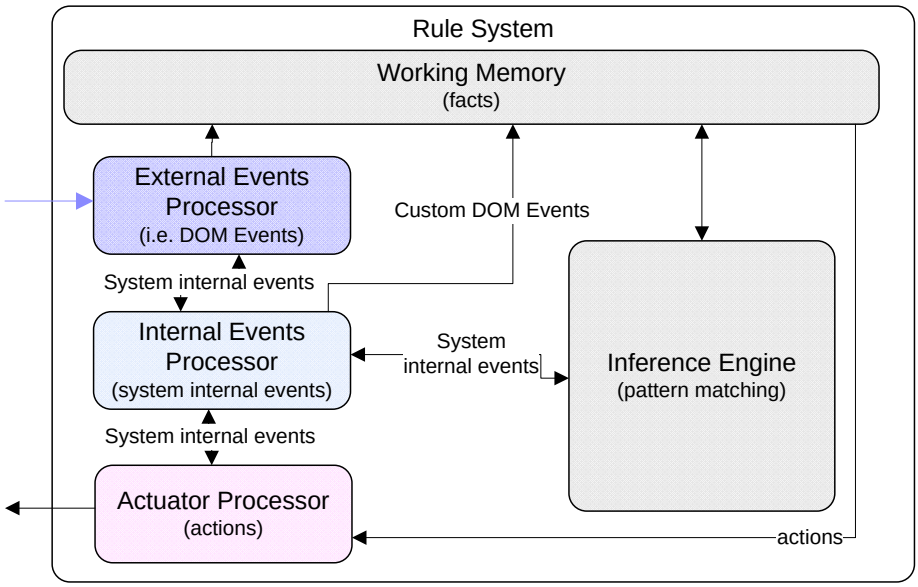


Figure 6.1: Mashups Engine - The Rule System

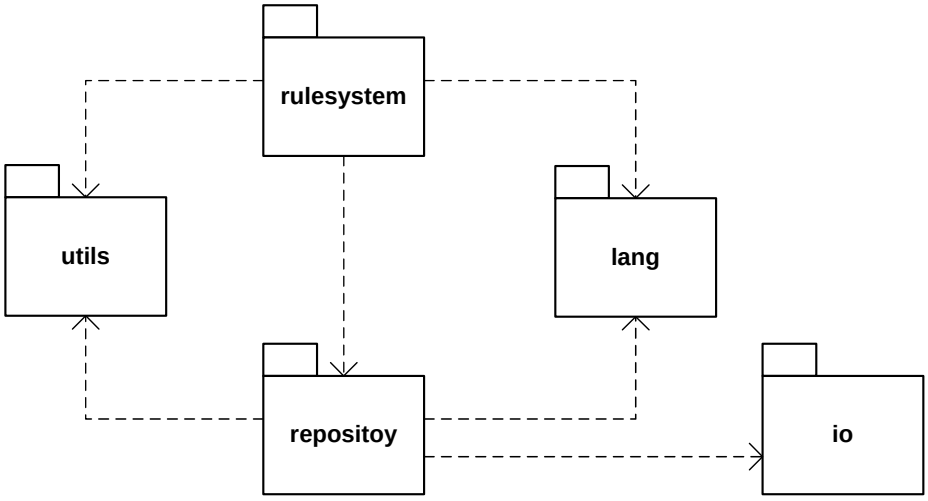


Figure 6.2: Main packages

contains the necessary code to load and parse the user defined plan (mashup description that end users have created). The repository uses code from the `io` (input / output) package to read the contents of the user defined plan. Default `io` implementation is AJAX based. The `lang` package contains the classes necessary to instantiate the execution language constructs. Hence the code in the repository package will first parse the contents and will then instantiate rules (create objects) using the classes in the lang package. The main parser that we are using is a JSON parser. However there are specific situations related to variables were we use regular expressions in the parsing process. For example, we are using such an approach based on regular expressions when we area dealing with a JavaScriptBooleanConditionalElement that is using variables. Hence we search for variables' names using the following regular expression `\\$\\w+`. Variable names are preceded by the sign $. The utils package contains a series of utilities classes. These classes implement for example algorithms for arrays management and so forth.

The `rulesystem` package comprises a series of sub-packages as seen in Figure 6.3. These packages are an exact mirror of the conceptual components depicted in Figure 6.1. The `eventprocessors` package contains both the external and internal event processors. As a general rule, the oriented associations lines between the packages basically express dependencies. The external event processor uses a default list of events on which it listens for. The default list of events contains the W3C DOM Events. W3C DOM Events are organized in several major categories: `BasicEvents`, `KeyboardEvents`, `MouseEvents`, `MouseWheelEvents`, `MouseMultiWheelEvents`, `MutationEvents`, `MutationNameEven` `UIEvents`.

The entire implementation is written in JavaScript. Hence the implementation of such a rule based system for execution is very different than a regular implementation in Java for example, especially because JavaScript is weakly typed and because of the environment where the code is going to run is very demanding in terms of memory consumption. Because of the memory issues, as a general guideline, we argue that iterative implementations should take priority over recursive implementations.

Although there are today a large number of JavaScript toolkits (i.e. jQuery [1], MochiKit [2], Proto-

[1] http://jquery.com/

[2] http://mochi.github.io/mochikit/

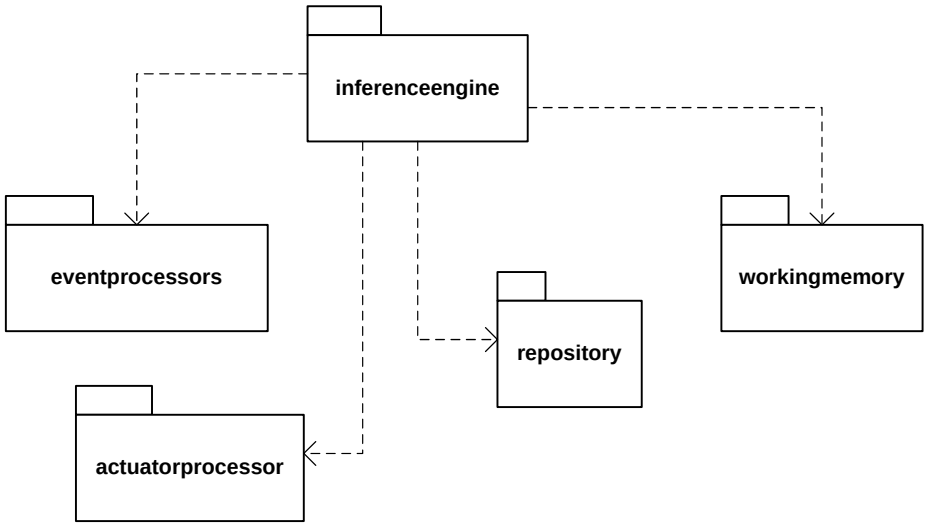


Figure 6.3: Rule System packages

type+Scriptaculous [3] and so forth) that can speed up JavaScript development we have opted out for Dojotoolkit [4]. We have chosen dojo because:

- dojotoolkit provides fundamental development constructs that are similar to OOP and especially to Java;
- it is professional development oriented;
- provides a full stack for project management, build system, testing and so forth;
- it is corporate sustained: IBM and Sitepen.

As depicted in Figure 6.4 our engine can have access to the entire DOM structure, styles (CSS) and events of a web page (service) or several services depending on where the engine resides. There are two possibilities: either the engine is used at the level of a web page and then the engine has access only to content that is strictly related to web page that loaded the engine. One should know that because of security reasons if a web page contains `iframe` elements, the contents of these `iframes` are not accessible unless they load content from the same web server. The second possibility which gives full flexibility and allows creation of mashups is to use the engine at the browser level as an add-on of browser plugin. We are testing our prototype with Mozilla Firefox. Being an add-on then the engine has access to all content: browser tabs, services, and the browser itself if this is implemented using an XML derivative as we already argued in the preceding chapter. The engine requires as input the user defined plan (mashup). It is worth mentioning that web sites can request the existence of a particular add-on in order to function properly.

The implementation has been developed taken into consideration the environment, memory limitations and the fact that users can not be kept waiting for an answer too long. Hence for instance the transitive closure algorithm uses only lists of objects' indexes and not lists containing entire objects. Thus in the context of our recurrent conferences calendar use case, the inference process of an early

[3] http://script.aculo.us/
[4] http://dojotoolkit.org/

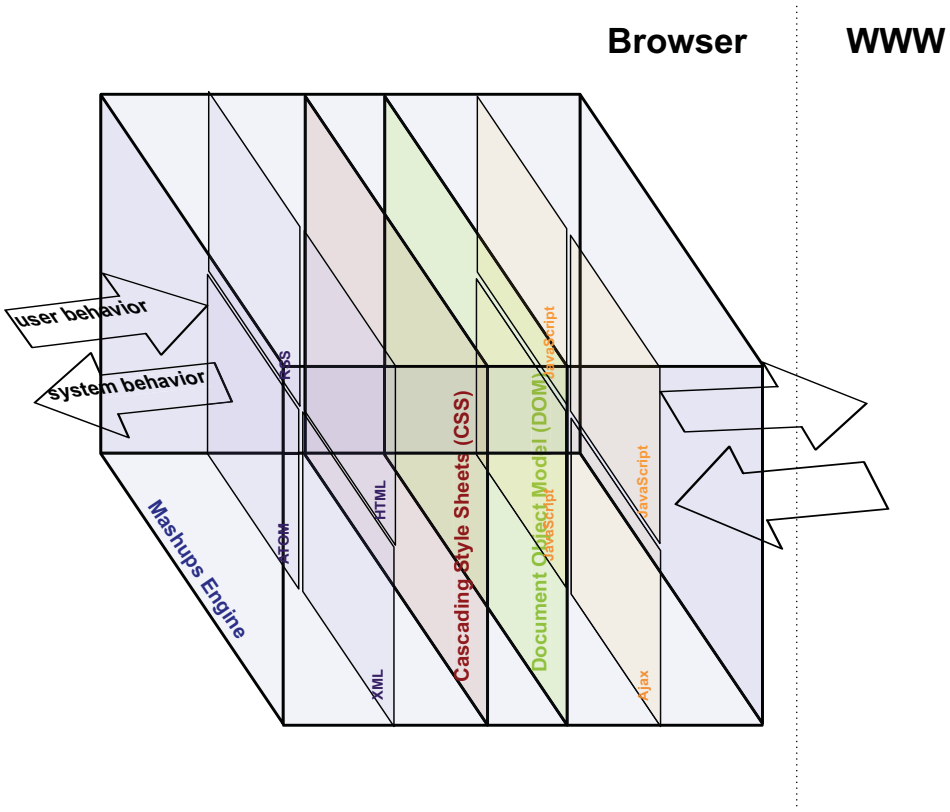


Figure 6.4: Browser Layers

stage implementation took in terms of time more than 1 minute. Current implementation that uses an algorithm based on indexes takes less than a second. We find important to mention that the dbworld web page contains more than 10000 DOM nodes.

An early stage prototype that is capable of running very basic rules (among other things is missing `forall` existential implementation) and capable of addressing simple use cases such as basic web site personalization has been made available as a Google code project - https://code.google.com/p/jsonrules/. Figure 6.5 shows a Google Analytics map of visits concerning the period of time between August 2009 when we have uploaded the project to February 9, 2014. The top 6 countries by the number of visits where (United States of America: 2217 , India: 458, Germany: 368, United Kingdom: 274, France: 215 and Canada: 203).

6.2 Use Cases

We discuss in this section a series of use cases that address different topics in order to emphasize the generality of our approach. The approach and the execution is unique so we will not insist on the process itself but on the use cases themselves.

6.2.1 Conference Calendar - The Recurrent Use Case

The Conferences Calendar is our recurrent example that we used through out this thesis. The outcome of this use case is to help end-users to store in semi-automatic way conferences announced on DBWorld

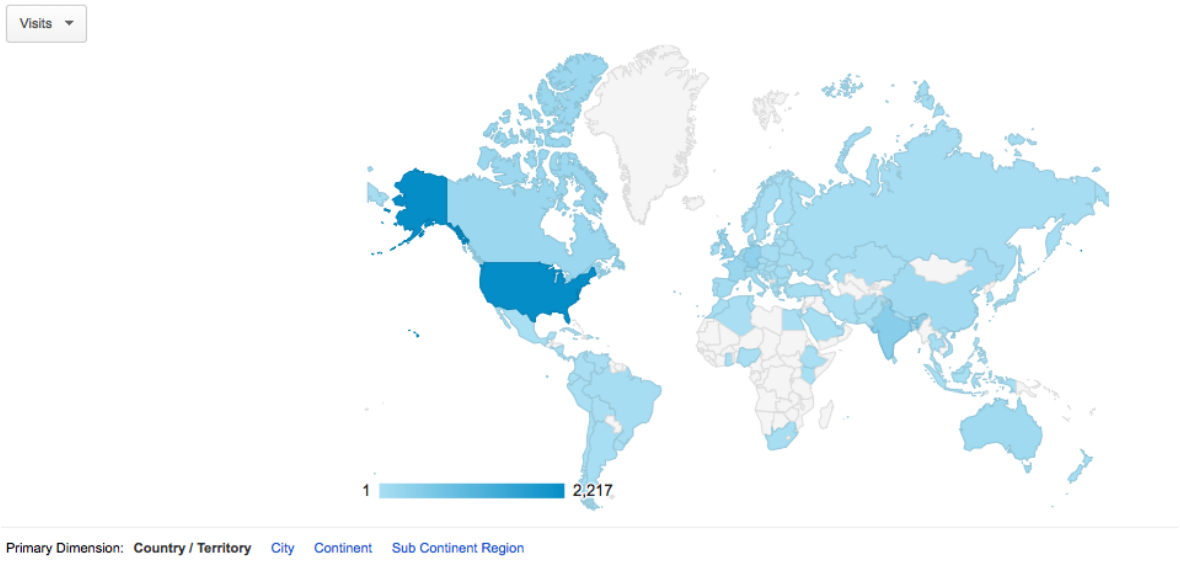


Figure 6.5: Google Analytics Map of Visits (Aug 2009 - Feb 2014)



Sent	Message Type	From	Subject	Deadline	Web Page
10-Feb-2014	journal CFP	Silviu Maniu	DAPD Journal Special Issue on Databases and Crowdsourcing -- deadline extended	28-Feb-2000	web page
10-Feb-2014	conf. ann.	Chang Liu	CFP: IEEE BDSE2014 (Big Data Science and Engineering), Beijing, China, 24-26 Sept. 2014	5-May-2014	web page
10-Feb-2014	conf. ann.	Yu Cao	CALL FOR PAPERS, The 3rd ASE International Conference on Big Data Science and Computing (BigDataScie	30-Apr-2014	web page
10-Feb-2014	conf. ann.	Yu Cao	Call for Workshop Proposals for 3rd ASE Conference on Big Data Science and Computing	22-Feb-2014	web page
10-Feb-2014	conf. ann.	Masao Mori	Call for Papers:5th International Conference on E-Service and Knowledge Management	28-Feb-2014	web page

Figure 6.6: DbWorld Service

mailing list in a Google Calendar. We say semi-automatic because each user is looking for a different conference, hence before storing them an end-user needs to find them and select them. Now to achieving this is a time consuming and heavy process. First of all DbWorld (see Figure 6.6) does not offer a search functionality, and therefore you are obliged to use the search functionality offered by the web browser or just scroll up and down until you find all what you are looking for (not a good option). Second, one needs to go back and forth between to browser tabs, one for DbWorld and one for Google Calendar. For each event one needs to copy and paste at least twice (subject of the event and due date).

There are a series of problems that we would like to resolve with one app:

- see everything in just one tab, avoiding going back and forth between open tabs;
- provide a way to search in DbWorld list other than using the browser search;
- allow to analyze the search results before storing them in the calendar;
- select conference and store them automatically in the calendar.



Figure 6.7: Calendconf Stand Alone Service

To resolve these issues we created first a very basic web page (service which we called Calendconf) with just an input field and a search button which we will use together with a user defined plan (mashup) and the DbWorld service to perform search. Each entry found will be added in Calendconf to be further analyzed before being stored in Google Calendar. Figure 6.7 presents the stand alone Calendconf service. It looks like any other web page (service). The difference is that as mentioned earlier this service asks for a browser add-on (the engine) to be installed, otherwise the service will not work. After the installation of the add-on the contents of the browser tab change dramatically (see Figure 6.8). After the engine has been installed and provided with the user defined plan, one tab browser contains all the services described in the user defined plan.

According to our conceptual model which we discussed in Chapter 3 we are dealing here with 3 main contexts. Each of these contexts is related to a service: Calendconf, DbWorld and Google Calendar. Again as discussed in the same chapter a context is identified by a set of concepts that are comprised in these contexts. Hence for example for the Calendconf context we have the service identified by an URL (i.e. http://calendconf.localhost/), and the search input and search button. Each of the latter concepts (search input and search button) have a series of characteristics that allow one to identify them. For instance the HTML representation of the search input field is `<input id="mashSearchInput" type="text" size="50" name="mashSearchInput"></input>`. As such we consider enough to use the `type` which is an `INPUT` and the `name` attribute which has the value `mashSearchInput` to uniquely identify this search field. All concepts that we need or use in rules are identified in a similar manner.

Figure 6.9 depicts the search and save operations. We have searched for "DEXA Workshop" and the results are added for review in the Calendconf service under the search input field. The image also shows that the first entry has been checked for save. In the Google calendar the event has been saved. The Event contains the url of message on DbWorld and the deadline date. We have used for this use 6 rules: (1) one to clean up / delete search results before doing a new search; (2) one to perform the search in the DbWorld service; (3) and 3 rules to do the save of the event in Google calendar.

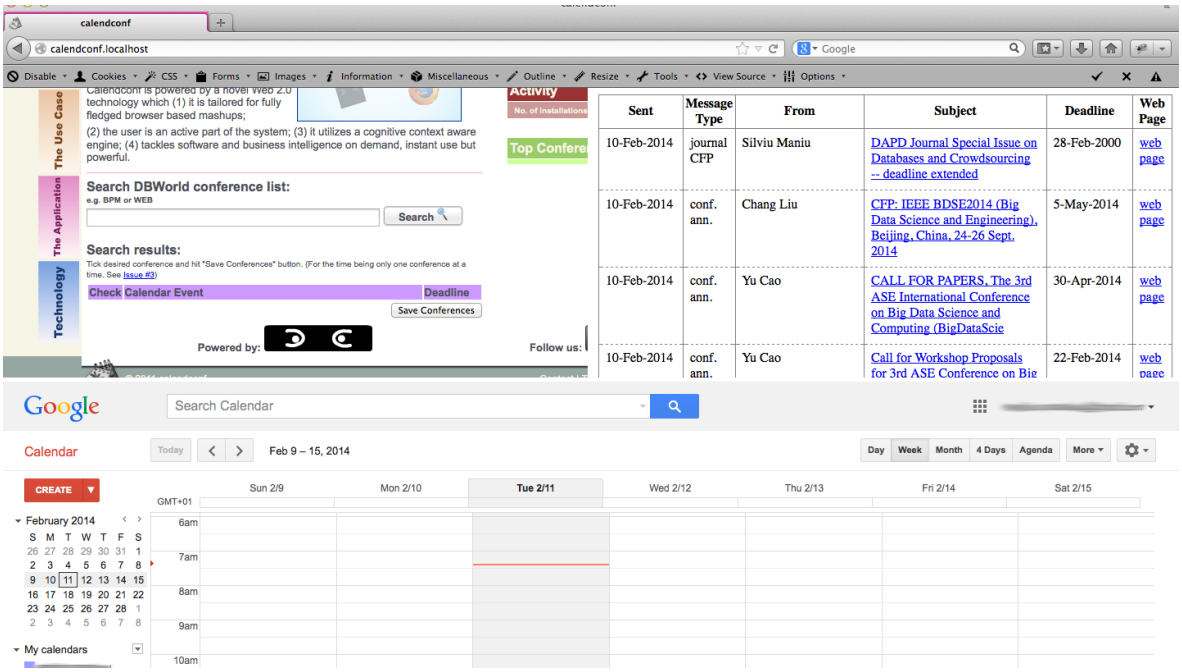


Figure 6.8: Calendconf Mashup

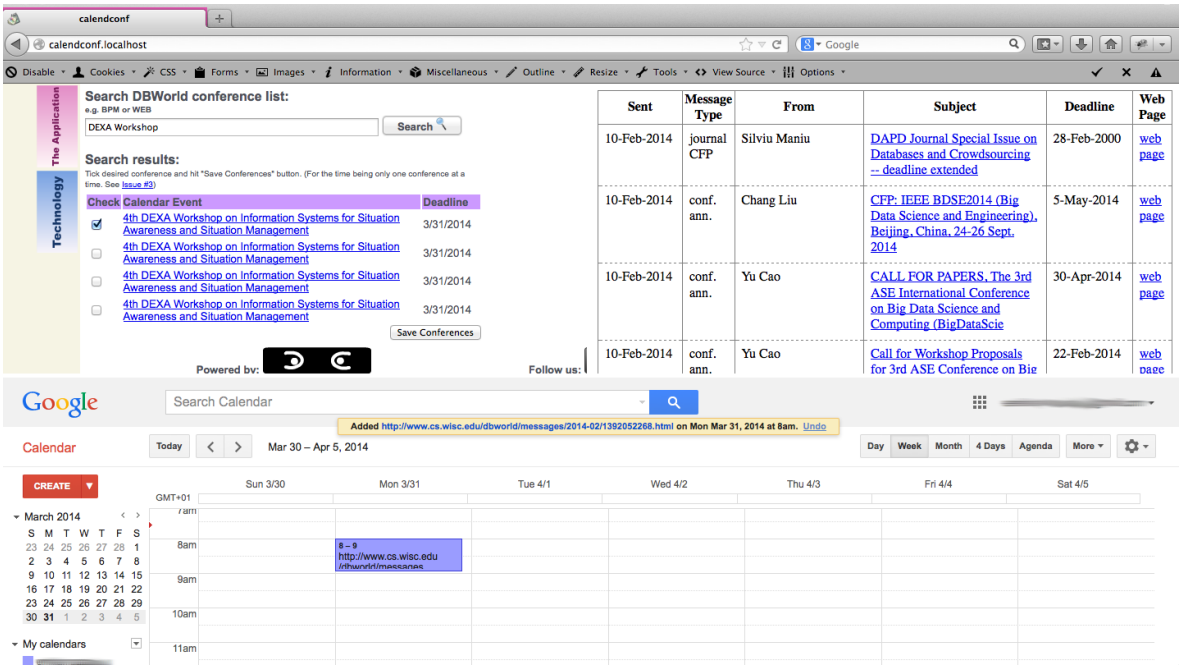


Figure 6.9: Calendconf Mashup - Search and Save

6.2.2 Security - Web Policies

This use case is based on [Iannella, 2009]. The author argues that there is a move towards Policy-Oriented Web because of e-Society communities. With so much content generated by the end-users, and shared over these social networks, "there is the real danger that the implicit sharing rules that communities have developed over time will be lost in translation in the new digital communities" [Iannella, 2009]. We subscribe to the opinion of the author. Social networks like FaceBook have been quite successful because they have provided features that empower end-user to express themselves online, mainly by sharing content with friends and colleagues. However this social online experience can have serious repercussions is the rules under which content is shared is not known or even worse not respected.

The simplest example is that on FaceBook we can share photos of our friends. In addition we can tag these friends in the photos. However we do not know if for example our friends want to share their photos with anyone else but us. These type of restriction can be achieved to some extent, by specifying individual friends that can see a picture, or those who can not see a picture. Nonetheless at this point, when an user, either friend or not, sees your photo, they have the usual functionality that any web browser provides to "Save Image As" to the local disk. After this, the photo is out of your reach and is out of the FaceBook control as well.

Hence the question is how can we enforce or block this type of functionality when images are tagged. Figure 6.10 depicts such a tagged photo. Our approach can be used to addresses such cases. Hence end-user can define a rule or a set of rules that should be applied in this type of situations. However this will work only if the end-user has installed the engine and the user defined plan containing this rule or rules. But as we have stated in the previous section, web sites could ask users to install add-ons in order for the web site to work.

Blocking users to save the image to their disk could be achieved in several ways: (1) remove the "Save Image As" from the contextual menu - however this would be more difficult to achieve; (2) block the right click functionality and hence not displaying the context menu; and (3) add a transparent image over the real image - in this way the user will save actually a different image than the real one. Flicker for example provides a similar approach to block users from saving the images.

No matter which of these three actions is taken first what is required, is to identify that we are indeed in the correct context. Hence to achieve this we need to describe in the user defined plan the context that is about the FaceBook service. Plus the other concepts that are telling us we are dealing with an image that has been tagged, are as depicted in Figure 6.10 presents an example of such a tagged image. The right hand side of the page containing the image has a section that starts with the word `with`, followed by the FaceBook ids of the persons that are tagged in the image. Analyzing the structure of the page we find out that the tags reside in a `span` element that has as value for the attribute class the value `fbPhotoTagList`. The image is displayed in a `div` element with the value for the attribute `class` being `stage`. Therefore these two concepts in relationship with the service would be enough to identify that we are in the case of a tagged image on facebook and hence we would like to hide the image underneath a transparent div.

6.2.3 Personalization

Web site personalization is yet another type of use case that can be addressed with the approach we proposed in this thesis. Having the engine running as browser add-on would allow end-users to define user defined plans that concern any website. For instance when ever an user is reading an email in GMail, advertisements are displayed above the email subject (see Figure 6.11).

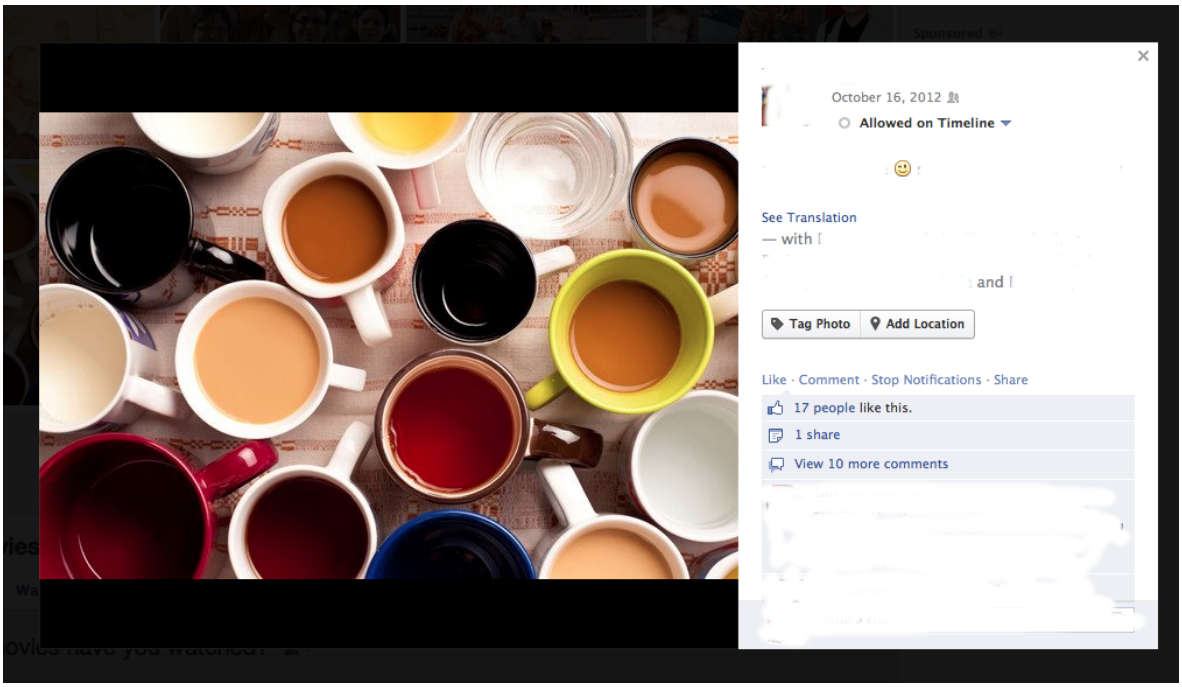


Figure 6.10: Facebook - Tagged Image Policy

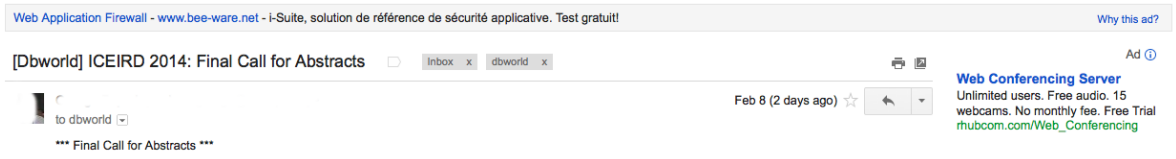


Figure 6.11: Personalization Use Case - Remove Gmail Add - Before Rule Execution

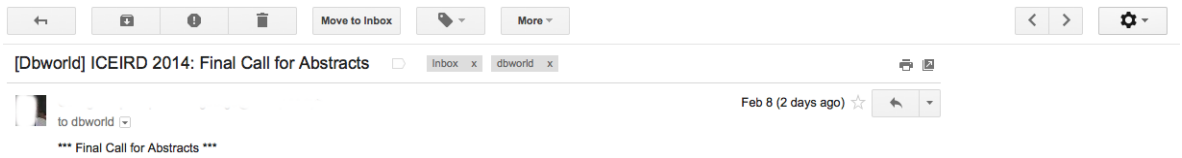


Figure 6.12: Personalization Use Case - Remove Gmail - After Rule Execution

Figure 6.11 depicts the visual representation of situation. This visual representation of the add above the mail subject is given by the following `<div class="mq"></div>`. Therefore in order to get rid of this advertisement a rule to do so would sound like: `ON event load IF exists an element of type div with class attribute equal to mq then delete this node`. The result of executing such a rule is depicted in Figure 6.12.

6.2.4 Web Analytics

Web site analytics are a well known topic of interest for the web. Platforms such as Google Analytics [5] or Piwik [6] are examples of tools that address this issue. Both of them offer almost the same insights, i.e. visits, downloads, keywords and so forth.

Our approach can be used straight forward for this type of use cases. Moreover specific rules can be defined to describe complex situations. For instance let's take the case of an online shop. With our approach it is easy to define a set of rules that for example transmit in real time, the activity of an user on the web site. Which are the articles that he/she is currently browsing, which were the articles that were added to the shopping cart but were later deleted from the shopping cart. Real time analytics can be used also to suggest on the fly new products based on what the user is viewing. The great advantage over the other approaches is that logging and things of interest are expressed by means of rules and processes. Hence if at some point in time, changes are required, this is very easy to achieve. Moreover because the engine uses directly the DOM and DOM Events we argue that the type of rules that can be described are limited only by the end-user's imagination. In addition because a larger and more complex set of information can be sent back home to be analyzed, the use of an intelligent document management such as FUI Polymathic [7] together with our engine can provide different and complex views on the collected data in relationship with business concepts that the company uses.

However when a service is used via a mashup in relationship with other services it is not possible to get information on how the web site (service) is used in relationship with other web sites (service). The mashup itself, just looking at the contexts defined, the services to which the described contexts refer to as well as based on the rules who are connecting these concepts together (please recall that for our conceptual model everything is a concept) can provide this type of statistics. In addition in such a situation, the provider of the mashup can gather statistics via our engine about other services again in relationship with the base service.

6.3 Requirements

We recapitulate in this section the list of requirements we introduced in Section 2.6 and explain how they have been fulfilled.

R1 such a system should allow evolution, sharing and distribution; end-users should be allowed through direct input to update / adapt the application;

Description: User defined plans (mashups) are in JSON format and stored basically in a file. However as depicted in Figure 4.7 in Chapter 4 they can be stored as well in a database and / or made available through a marketplace. Hence these mashups can be shared and distributed. Users can update and / or adapt them to their own needs and to other services.

R2 such a system should not be domain specific, and should allow a wide range of use-cases;

Description: Our engine uses directly the DOM and DOM Events which are platform and language neutral interfaces. In consequence our execution engine is platform and language neutral, and therefore is as general as these languages. Moreover we have already discussed a series of use cases to emphasize even more this aspect.

[5] http://www.google.com/analytics/
[6] http://piwik.org/
[7] http://polymathic.cnam.fr/

R3 in a such a system the end-user should be the coordinator of how the system works; hence the system should provide a new approach for describing behavior of composite systems (humans + services) (humans + services) [8];

Description: Users define mashups that the engine executes and follows. In addition these mashups describe also how the engine should behave based on the behavior that the end-users exhibit. In consequence we argue that indeed we have defined a new approach for describing behavior of composite systems: humans + services.

R4 such a system should support both skilled developers as well as novice users;

Description: Skilled developers can definitely use our approach, without any difficulties. The approach improves and provides means also for experienced developers to improve the process of writing code. We believe that we have improved and simplified things for novice users. However we accept that we have not managed to completely hide technology, since users are still required to write the mashups in a JSON format. To fully hide the technology, for example, a visual approach that uses pattern based interaction would be required. Nonetheless visually created model would still require an execution engine as the one discussed here. As such we argue that through this approach we are only missing the visual modeling link to fully achieve end-user development environment that is general enough.

R5 such a system should focus on the end-user and not on the system itself. The system should be hidden from the end-user as much as possible by providing the right level of representation such that a problem representation could be automatically translated into the core concepts of the underlying programming language in which the overall system is implemented;

Description: We believe that this requirement is already proven based on the previous statements.

R6 such a system should allow on demand development using the web as a platform or web as open application execution environment: build upon existing ideas, sites, applications [9];

Description: Because our system has been developed as a component of a web browser which already can access the web, we can therefore get access to any of these services in a unified way via the browser.

R7 such a system should be compliant with SOA principles of: loose coupling, reusability, discoverability, composability;

Description: Again because we are using DOM and DOM events, we are not bound to any specific technology, and we can arbitrarily use any service that can be accessed via the web browser, in an unbounded manner. Parts of the user defined plans that refer to a particular service can be reused in relationship with other services. Services can be discovered as usually via regular approaches that are available in a web browser.

R8 such a system should allow decentralized and delocalized execution of software / components [10];

[8] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, retrieved 13 January 2014
[9] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/questions.pdf, retrieved 13 January 2014
[10] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, retrieved 13 January 2014

Description: As per our discussion in Section 4.4.3 this requirements is fulfilled.

R9 such systems should allow simultaneous build time, run time development and experiment [11].

Description: User defined plans are interpreted, hence the build time, run time and experiment happen almost in the same time.

[11] http://cordis.europa.eu/fp7/ict/ssai/docs/fp8-preparations/researchchallenges.pdf, retrieved 13 January 2014

Chapter 7

Conclusions

World Wide Web (WWW) has become the greatest repository of information that man has ever assembled and it is continuously growing. The new WWW or *Future Internet* is that of an Internet of Services and Internet of Things. Naturally, a series of questions arise from this context: how do you filter things to create more value than you currently get? how do you aggregate things in an intelligent and easy way instead of doing it in your head? The world cannot be described unambiguously, so how can you allow users to deal with the world in their own way, based on their understanding?

It has been argued extensively that the right solution comes from the right participants (the end-users). Unfortunately although a lot of effort has been put into developing a large number of frameworks to tackle some of the issues that this new environment has brought (one can see Chapter 2), design and deployment of such software capable of direct interaction and empowerment of the end-user is still an issue.

Our goal in this thesis was to address this lack of tools that are capable of direct interaction and empowerment of end-users, in a unified manner. To achieve this we proposed a conceptual user-centric approach.

To implement it we developed a conceptual model for this approach, we proposed an architecture that complies with the approach, and also proposed an execution engine that uses ECA rules and processes to execute applications that end-users have defined as mashups.

The systems that we envision has to allow a new programming model for composite systems (humans + services). We argue that such systems are intelligent systems being able to interact with the end-user according with an agreed beforehand plan, supporting evolution, sharing and distribution. Hence these systems are two layer systems: one high level layer, that deals with the problem at a conceptual and semantic level (the agreed on plan) and one low level layer that deals with the internals of the system and low level technologies i.e. direct access to services, etc. The low level layer should be hidden as much as possible from the end-user. The aspects that drive the development of our approach are: end-user oriented or user-centric; humans and system interacting with each other; plan; two-layer system; intelligent system.

To complement the approach we proposed a conceptual model. As UML is considered to be the *de facto* standard modeling language we also used it to formalize our conceptual framework. The main elements of our conceptual model are: Concept, Context, Behavior and Mashup. Instances of this conceptual model represent the user defined plans.

A web system that complies with our approach, to the best of our knowledge does not exists yet. However a GPS system does indeed complies with the conceptual approach we have introduced. Therefore our system architecture has been inspired from the GPS devices. However GPS devices

resolve a very specific problem, compared to wide range of applications that can be developed using the Web as a platform. From our point of view the most important aspects that we learn from the GPS devices are: first that, the low level layer of the system (system level) has been designed by default as real time system and second that at the system level there is a unified way of representing information. We argue that these two aspects provide the basement and are fundamental requirements for building systems that comply with the approach we introduced.

The execution engine that follows the approach and architecture is a web browser system. The engine is a variation of a production system. End-users define mashups (representing behavior related to contexts). These models are fed into the engine and they are executed by the engine. Both users and the system interact with each other via behavior that is expressed in the web browser.

We argue that with the approach discussed through out this thesis we have advanced a lot towards a fully user-centric approach that allows end-users to create their own applications using services oriented architectures.

From our point of view the missing link required to achieve a complete end-user system is a visual approach that will entirely hide all the few remaining aspects that still expose technical related aspects such as the mashup being written in a JSON format. In the next section we will shortly show future directions of research on the side of the subject and directions to achieve the missing visual modeling environment.

7.1 Future Work

We envision several possible directions for extending the work we presented and discussed in this thesis: (1) visual modeling of user defined plans; (2) usage if the approach with mobile platforms; (3) addition of different types of reasoning such as abductive logic programming.

7.1.1 Visual Modeling

We argue that visual modeling of user defined plan is the last missing link towards having a complete end-user approach that tackles the issues we discussed in this thesis. There are at least two directions for implementing a visual based solution for modeling user defined plans: (1) a solution that uses directly the UI and the services through interaction design patterns; (2) a solution that uses other established visual languages such as BPMN choreographies.

In order to improve end-user interaction with systems, over time a set of interaction design patterns have been defined [van Welie et Trætteberg, 2000]. Patterns based approaches seem also to improve understanding. There have been defined both a list of patterns [1] and list of design wizards [2] for the web context. These wizards concern specific web situations such as: selection and choice; data representation; navigation around; page types; page layout; page elements. We argue that a visual solution that takes advantage of these patterns and is capable of translating them into the mashup context in order to visually define *user defined plans* is worth investigating.

Choreographies refer to business-to-business collaborations and ensure interoperability between process orchestrations [Weske, 2007]. Choreographies are the specification of collaboration rules between businesses. BPMN 2.0 [OMG, 2009] specification defines a choregraphy also as a process, however it differs in purpose and behavior from a standard BPMN process. While a standard process or orchestration concerns the activities that are performed within a single business partner, process

[1] http://www.welie.com/patterns/

[2] http://patternwizard.nl/pattern/wizard/

choreogrpahies formalize interactions between business partners. In choreography terms a business partner refers to an organization. Hence the focus of a process choreography is on the message exchange between business partners. Especially because of this focus on exchanging messages and interactions between partners (services in the mashups case) we believe that choreographies can be used to model user defined plans.

7.1.2 Mobile Platforms

The proliferation of HTML 5 and JavaScript frameworks (see for instance Sencha Touch 2[3], PhoneGap[4], jQuery Mobile[5]) for building cross-platform mobile apps is a good enough incentive to try the approach and the prototype implementation we discussed in this thesis with these type of SDKs. We believe that the integration of our prototype with these type of SDKs should not pose too many difficulties.

The second perspective concerning mobile platforms is the use of our approach with the Android operating system. We argue that Android OS complies with the two fundamental aspects that we identified in Section 4.2.3 as being mandatory for implementing our approach. Hence being real time and having the conceptual model behind Android OS that includes concepts such as Activities, Intents, Services, Events. Therefore we believe that our approach fits very well with the Android OS and this perspective should be explored more.

7.1.3 New Reasoning Techniques

The reasoning engine is currently basically a forward chaining one. However the architecture allows extensions. Therefore further extensions can address the use of other reasoning techniques such as backward chaining, reasoning by abduction and so forth. Abductive logic programming could be particularly of interest because it allows some predicates to be incompletely defined. It would be useful to have this type reasoning techniques in the web environment and especially in an user-centric approach.

[3] http://www.sencha.com/products/touch/
[4] http://phonegap.com/
[5] http://jquerymobile.com/

Bibliography

[Abdelnur et Hepper, 2003] Alejandro Abdelnur et Stefan Hepper. JavaTM Portlet Specification, Version 1.0. JSR 168: Portlet Specification, October 2003. *http://jcp.org/en/jsr/detail?id=168*.

[Abiteboul *et al.*, 2008] Serge Abiteboul, Ohad Greenshpan, et Tova Milo. Modeling the mashup space. In *WIDM'08: Proceeding of the 10th ACM workshop on Web information and data management*, pages 87–94, 2008.

[Ackoff, 1971] R.L. Ackoff. Towards a System of System Concepts. *Management Science*, 17(11), 1971.

[Aghaee *et al.*, 2012] Saeed Aghaee, Marcin Nowak, et Cesare Pautasso. Reusable decision space for mashup tool design. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12)*, pages 211–220. ACM, 2012.

[Aghaee *et al.*, 2013] Saeed Aghaee, Cesare Pautasso, et Antonella De Angeli. Natural end-user development of web mashups. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2013)*, 2013.

[Aghaee et Pautasso, 2010] Saeed Aghaee et Cesare Pautasso. Mashup development with html5. In *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups*. ACM, 2010.

[Aghaee et Pautasso, 2013] Saeed Aghaee et Cesare Pautasso. Guidelines for efficient and effective end-user development of mashups. In Yvonne Dittrich, Margaret Burnett, Anders Mørch, et David Redmiles, editors, *Proceedings of 4th International Symposium, IS-EUD 2013*, volume 7897 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2013.

[Alba *et al.*, 2008] Alfredo Alba, Varun Bhagwan, et Tyrone Grandison. Accessing the Deep Web: When Good Ideas Go Bad. In *OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 815–818, New York, NY, USA, 2008. ACM.

[Alliance, 2009] Open Mashup Alliance. Enterprise Mashup Markup Language (EMML) . Open mashup Alliance Recommendation, 2009. *http://www.openmashup.org/omadocs/v1.0/index.html*.

[Altinel *et al.*, 2007] Mehmet Altinel, Paul Brown, Susan Cline, Rajesh Kartha, Eric Louie, Volker Markl, Louis Mau, Yip-Hing Ng, David Simmen, et Ashutosh Singh. Damia - A Data Mashup Fabric for Intranet Applications. In *Proceedings of the 33th International Conference on Very Large Data Bases*. VLDB, 2007. *http://www.vldb.org/conf/2007/papers/demo/p1370-altinel.pdf*.

[Analyti *et al.*, 2007] Anastasia Analyti, Manos Theodorakis, Nicolas Spyratos, et Panos Constantopoulos. Contextualization as an independent abstraction mechanism for conceptual modeling . *Information Systems*, 32(1):24–60, 2007.

[Auer et Lehmann, 2007] Sören Auer et Jens Lehmann. What Have Innsbruck and Leipzig in Common? Extracting Semantics from Wiki Content. In *ESWC*, pages 503–517, 2007.

[Baker et Dobson, 2005] Sean Baker et Simon Dobson. Comparing Service-Oriented and Dsitributed Object Architectures. In *Proceedings of the International Symposium on Distributed Objects and Applications*, number 3760 in LNCS, pages 631–645. Springer Verlag, 2005.

[Berglund *et al.*, 2007] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, et J. Simeon. *XML Path Language (XPath) 2.0*. W3C, 2 edition, January 2007. *http://www.w3.org/TR/xpath20/*.

[Bioernstad et Pautasso, 2007] B. Bioernstad et C. Pautasso. Let it flow: Building Mashups with Data Processing Pipelines. In *Proc. of Mashups'07 International Workshop on Web APIs and Services Mashups at ICSOC'07*, number 4907 in LNCS, pages 15–28, 2007. *http://www.jopera.org/files/jopera_mashup07.pdf*.

[Bizer *et al.*, 2010] Christian Bizer, Tom Heath, et Tim berners Lee. Linked Data - The Story So Far. *International Journal on Semnatic Web and Information Systems*, 2010.

[Bolchini *et al.*, 2007] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, et Letizia Tanca. A data-oriented survey of context models. *ACM SIGMOD Record*, 36(4):19–26, 2007.

[Booth *et al.*, 2004] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, et David Orchard. Web Services Architecture. *http://www.w3.org/TR/ws-arch/*, Feb 2004.

[Bos *et al.*, 2010] Bert Bos, Tantek Celik, Ian Hickson, et Hakon Wium Lie. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C Working Draft, December 2010. *http://www.w3.org/TR/CSS2*.

[Bray *et al.*, 2008] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, et Francois Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, November 2008. *http://www.w3.org/TR/2008/REC-xml-20081126/*.

[C. Lovelock, 1996] S. Vandermerwe et al. C. Lovelock, editor. *Services Marketing*, chapter 12. Englewood Cliffs, NJ: Prentice Hall, 1996.

[Caceres, 2009] Marcos Caceres. Widgets 1.0: Packaging and Configuration. W3C Candidate Recommendation, July 2009. *http://www.w3.org/TR/widgets/*.

[Cappiello *et al.*, 2011] Cinzia Cappiello, Florian Daniel, Maristella Matera, Matteo Picozzi, et Michael Weiss. Enabling end user development through mashups: Requirements, abstractions and innovation toolkits. In *Proceedings of the 3rd International Symposium, IS-EUD 2011, Torre Canne (BR), Italy, June 7-10, 2011*, volume 6654 of *LNCS*, pages 9–24. Springer, 2011.

[Cattell, 1994] R. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann Publishers, 1994.

[Chinnici *et al.*, 2007] R. Chinnici, J.J. Moreau, A. Ryman, et Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C, 2 edition, June 2007. *http://www.w3.org/TR/wsdl20/*, retrieved Novemeber 2008.

[Christensen *et al.*, 2001] Erik Christensen, Francisco Curbera, Greg Meredith, et Sanjiva Weerawarana. Web Services Description Language (WSDL). *http://www.w3.org/TR/wsdl*, Mar 2001.

[Chul *et al.*, 2009] Michael Chul, Andy Miller, et Roger P. Roberts. Six Ways to make Web 2.0 work. *Business Technology, The McKinsey Quaterly*, pages 1–6, February 2009.

[Clark et Sasse, 1997] Louise Clark et M. Angela Sasse. Conceptual design reconsidered: The case of the internet session directory tool. In *In Proceedings of HCI'97 People and Computers XII*, 1997.

[Collins, 2008] Stephen Collins. Enterprise 2.0 A new Age of Aquarius? http://www.acidlabs.org/2008/11/20/enterprise-20-a-new-age-of-aquarius/, November 2008.

[Coutaz *et al.*, 2005] Joelle Coutaz, James L. Crowley, Simon Dobson, et David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[Daniel *et al.*, 2009] Florian Daniel, Fabio Casati, Boualem Benatallah, et Ming-Chien Shan. Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In *Proceeding of the 2nd international workshop on Ontologies and Information systems for the Semantic Web (ER'09)*, volume 5829/2009 of *LNCS*, pages 428–443. Springer Berlin / Heidelberg, 2009.

[Davies *et al.*, 2009] J. Davies, M. Potter, M. Richardson, S. Stincic, J. Domingue, C. Pedrinaci, D. Fensel, et R. Gonzalez-Cabero. Towards the open service web . *BT Technology Journal*, 26(2):1694–1719, 2009.

[de Sainte Marie *et al.*, 2009] Christian de Sainte Marie, Adrian Paschke, et Gary Hallmark. RIF Production Rule Dialect. W3C Candidate Recommendation, October 2009. *http://www.w3.org/TR/rif-prd/*.

[Dey et Abowd, 1999a] Anind K. Dey et Gregory D. Abowd. Towards a better understanding of context and context-awareness. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, 1999.

[Dey et Abowd, 1999b] Anind K. Dey et Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[Easterbrook *et al.*, 2008] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, et Daniela Damian. *Guide to Advanced Empirical Software Engineering*, chapter Selecting Empirical Methods for Software Engineering Research, pages 285–311. Springer, 2008.

[Ennals *et al.*, 2007] Rob Ennals, Eric Brewer, Minos Garofalakis, Michael Shadle, et Prashant Gandhi. Intel mash maker: Join the web. *SIGMOD Record*, 36(4), 2007.

[Fielding, 2000] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[Fischer et Giaccardi, 2004] Gerhard Fischer et Elisa Giaccardi. *End User Development - Empowering People to Flexibly Employ Advanced Information and Communication Technology*, chapter Meta-Design: A Framework fo the Future of the End-User Development. Kluwer Academic Publishers, 2004.

[Fischer, 2012] Gerhard Fischer. Context-aware systems - the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way, to the 'right' person. In *AVI'12*. ACM, 2012.

[Forgy, 1982] Charles Forgy. Rete – A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.

[Foster et Tuecke, 2005] I. Foster et S. Tuecke. Describing the Elephant: The Different Faces of IT as Service. *Enterprise Distributed Computing*, 3(6):26–34, July/August 2005.

[free encyclopedia, 2009] Wikipedia-The free encyclopedia. Web widget. *http://en.wikipedia.org/wiki/Web_widget*, 2009.

[Freed et Borenstein, 1996] N. Freed et N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046 (Draft Standard), November 1996. Updated by RFCs 2646, 3798, 5147.

[Garrett, 2005] Jesse James Garrett. Ajax: A New Approach to Web Applications. *http://www.adaptivepath.com/ideas/essays/archives/000385.php*, Feb 2005.

[Google, 2009] Google. *gadgets.* API Developer's Guide*. Google, 2009. *http://code.google.com/apis/gadgets/docs/dev_guide.html*.

[Grammel et Storey, 2008] Lars Grammel et Margaret-Anne Storey. An End User perspective on Mashup Makers. Technical report, University of Victoria, September 2008. *http://lars.grammel.googlepages.com/paper_mashup_makers.pdf*.

[Gregorio et de hOra, 2007] J. Gregorio et B. de hOra. The Atom Publishing Protocol (RFC5023). *http://tools.ietf.org/html/rfc5023*, 2007.

[Group, 2009] W3C OWL Working Group. OWL 2 Web Ontology Language. http://www.w3.org/TR/owl2-overview/, 2009.

[Grudin, 2001] Jonathan Grudin. Desituating action: digital representation of context. *Human-Computer Interaction*, 16(2):269–286, 2001.

[Guarino, 1998] Nicola Guarino. Formal Ontology and Information Systems. In *Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998*, pages 3–15. IOS Press, Amsterdam, 1998.

[Gudgin *et al.*, 2007] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, et Yves Lafon. SOAP. *http://www.w3.org/TR/soap12-part1/*, Apr 2007.

[Guizzardi, 2005] Giancarlo Guizzardi. *Ontological Foundations for Structural Conceptual Models*. PhD thesis, Telematics Instituut, Enschede, The Netherlands, 2005. Telematica Instituut Fundamental Research Series No. 15, ISBN 90-75176-81-3.

[Guo *et al.*, 2008] Rui Guo, Bin Zhu, Min FENG, Aimin PAN, et Bosheng ZHOU. Compoweb: a component-oriented web architecture. *WWW '08: Proceeding of the 17th international conference on World Wide Web*, Apr 2008.

[Hadley, 2006] M. J. Hadley. *Web Application Description Language (WADL)*. Sun Microsystems Inc., November 2006. *https://wadl.dev.java.net/wadl20061109.pdf*, retrieved Novemeber 2008.

[Hanson et Hasan, 1993] E. Hanson et M. Hasan. Gator: An optimized discrimination network for active database rule condition testing. Technical report, Univ. of Florida, 1993.

[Harris, 2013] Derrick Harris. A peek inside microsoft's new 'design-first' development strategy. *Gigaom*, October 2013. accessed on 21 October 2013.

[Hately *et al.*, 2004] L. C. A. Hately, C. von Riegen, et T. Rogers. Uddi specification version 3.0.2. Technical report, OASIS, 2004.

[Hepper, 2008] Stefan Hepper. JavaTM Portlet Specification, Version 2.0. JSR 286: Portlet Specification 2.0, June 2008. *http://jcp.org/en/jsr/detail?id=286*.

[Hohpe et Woolf, 2003] Gregor Hohpe et Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Hors *et al.*, 2004] A. Le Hors, P. Le Hegaret, L. Wood, G. Nicol, J. Robie, M. Champion, et S. Byrne. Document Object Model (DOM) Level 3 Core Specification. W3C Recommendation, April 2004. *http://www.w3.org/TR/DOM-Level-3-Core/*.

[Hoyer *et al.*, 2009] Volker Hoyer, Florian Gilles, Till Janner, et Katarina Stanoevska-Slabeva. Sap research rooftop marketplace: Putting a face on service-oriented architectures. In *SERVICES '09: Proceedings of the 2009 Congress on Services - I*, pages 107–114, Washington, DC, USA, 2009. IEEE Computer Society.

[Iannella, 2009] Renato Iannella. Towards e-society policy interoperability. In *Proceedings of the 9th IFIP WG 6.1 Conference on e-Business, e-Services and e-Society, I3E 2009, Nancy, France, September 23-25, 2009*, volume 305 of *IFIP Advances in Information and Communication Technology*, pages 369–384. Springer, 2009.

[IBM, 2008] IBM. *IBM InfoSphere MashupHub. User and Administrator Guide*. IBM, 1 edition, 2008.

[Jackson et Wang, 2007] Collin Jackson et Helen J. Wang. Subspace: Secure Cross-domain Communication for Web Mashups. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 611–620, New York, NY, USA, 2007. ACM.

[Janner *et al.*, 2009] Till Janner, Robert Siebeck, Christoph Schroth, et Volker Hoyer. Patterns for enterprise mashups in b2b collaborations to foster lightweight composition and end user development. *ICWS '09: IEEE International conference on Web Services 2009*, 0:976–983, 2009.

[Jarrar et Dikaiakos, 2008] M. Jarrar et M. D. Dikaiakos. Mashql: a query-by-diagram topping sparql. In *ONISW '08: Proceeding of the 2nd international workshop on Ontologies and nformation systems for the semantic web*, pages 89–96, New York, NY, USA, 2008. ACM.

[Kagermann, 2008] H. Kagermann. Toward a European Strategy for the Future Internet A Call for Action. White paper, SAP AG, 2008. *http://www.sap.com/about/company/research/fields/internet_services/index.epx*.

[Keukelaere *et al.*, 2008] Frederik De Keukelaere, Sumeer Bhola, Michael Steiner, Suresh Chari, et Sachiko Yoshihama. SMash: Secure Component Model for Cross-Domain Mashups on Unmodified Browsers. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 535–544, New York, NY, USA, 2008. ACM.

[Klyne et Caroll, 2004] G. Klyne et J.J. Caroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, February 2004. *http://www.w3.org/TR/rdf-concepts/*.

[Ko *et al.*, 2011] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan F. Blackwell, Margaret M. Burnett, Martin Erwig, Christopher Scaffidi, Joseph Lawrance, Henry Lieberman, Brad A. Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, et Susan Wiedenbeck. The state of the art in end-user software engineering. *ACM Computing Surveys*, 43(3):1 – 44, 2011.

[Kunze, 2009] Matthias Kunze. Business Process Mashups – An Analysis of Mashups and their Value Proposition for Business Process Management. Master's thesis, Hasso Plattner Institut an der Universität Potsdam, 2009.

[Laga *et al.*, 2009] Nassim Laga, Emmanuel Bertin, et Noel Crespi. A web based framework for rapid integration of enterprise applications. *ICPS '09: Proceedings of the 2009 international conference on Pervasive services*, Jul 2009.

[Levine, 2009] Rick Levine. *The Cluetrain Manifesto*. Basic Books, 10th edition, 2009.

[Lieberman *et al.*, 2006] Henry Lieberman, Fabio Paterno, et Volker Wulf, editors. *End-User Development*, volume 9 of *Human-Computer Interaction Series*. Springer, 2006.

[MacKenzie *et al.*, 2006] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, et Rebekah Metz. OASIS Reference Model for Service Oriented Architecture. *http://docs.oasis-open.org/soa-rm/v1.0/*, Oct 2006.

[Maximilien *et al.*, 2007] E. Michael Maximilien, Hernan Wilkinson, Nirmit Desai, et Stefan Tai. A Domain-Specific language for Web APIs and Services Mashups. In *Proceedings of ICSOC 2007*, volume 4749 of *LNCS*, pages 13–26. Springer-Verlag Berlin Heidelberg, 2007.

[McDaniel, 2003] Scott McDaniel. What's your idea of a mental model? *http://boxesandarrows.com/whats-your-idea-of-a-mental-model/*, February 2003. Retrieved 11th Janury 2014.

[Menand, 1997] Louis Menand. *Pragmatism: A Reader*. Vintage, 1997.

[Miranker, 1987] D. Miranker. Treat: A better match algorithm for AI production systems. In *Proceedings of the AAAI'87 Conference*, 1987.

[Mockapetris, 1987] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592.

[Morbidoni *et al.*, 2007] Christian Morbidoni, Axel Polleres, Danh Le Phuoc, et Giovanni Tummarello. Semantic Web Pipes. Technical report, DERI, November 2007.

[Nardi, 1993] Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, 1993.

[Newell, 1994] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1994.

[Nottingham et Sayre, 2005] M. Nottingham et R. Sayre. The Atom Syndication Format (RFC4287). *http://tools.ietf.org/html/rfc4287*, 2005.

[Ogrinz, 2009] Michael Ogrinz. *Mashup Patterns: Design and Examples for the Modern Enterprise*. Addison-Wesley Professional, 2009. ISBN:978-0321579478.

[OMG, 2007] OMG. Production Rule Representation (PRR), Beta 1, November 2007.

[OMG, 2008] OMG. Semantics of Business Vocabulary and Business Rules Specification. http://www.omg.org/spec/SBVR/, January 2008.

[OMG, 2009] OMG. Business Process Model and Notation (BPMN). FTF Beta 1 for Version 2.0. http://www.omg.org/spec/BPMN/2.0, August 2009.

[OpenSocial, 2009] OpenSocial. *OpenSocial Gadgets API Specification v0.9*. OpenSocial, 2009. *http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/Gadgets-API-Specification.html*.

[O'Reilly, 2005] T. O'Reilly. What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software. *Oreillynet.com*, September 2005. *http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html*.

[O'Reilly, 2007] T. O'Reilly. What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software. *Communications and Startegies*, 1st quarter(65):17, 2007.

[Pascalau et Giurca, 2009a] Emilian Pascalau et Adrain Giurca. A Lightweight Architecture of an ECA Rule Engine for Web Browsers. In *Proceedings of 5th Knowledge Engineering and Software Engineering, KESE 2009, collocated with KI 2009*, volume 486, pages 9–20. CEUR Workshop Proceedings, 2009.

[Pascalau et Giurca, 2009b] Emilian Pascalau et Adrian Giurca. A Rule-Based Approach of Creating and Executing Mashups. In C. Godart et al., editor, *Proceedings of the 9th IFIP Conference on e-Business, e-Services, and e-Society (I3E 2009)*, volume 305 of *IFIP AICT*, pages 82–95. Springer, 2009.

[Pascalau et Rath, 2010] Emilian Pascalau et Clemens Rath. Managing Business Process Variants at eBay. In Jan Mendling et Mathias Weske, editors, *Proceedings of the 2nd International Workshop on BPMN, BPMN2010*. Springer, 2010.

[Pascalau, 2011a] Emilian Pascalau. Mashups: Behavior in context(s). In *Proceedings of 7th Workshop on Knowledge Engineering and Software Engineering (KESE7) at the 14th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2011)*, volume 805, pages 29–39. CEUR-WS, 2011.

[Pascalau, 2011b] Emilian Pascalau. Towards TomTom like systems for the web: a novel architecture for browser-based mashups. In *Proceedings of the 2nd International Workshop on Business intelligencE and the WEB (BEWEB11)*, pages 44–47. ACM New York, NY, USA, 2011.

[Pedrinaci et Domingue, 2010] Carlos Pedrinaci et John Domingue. Toward the next wave of services: Linked services for the web of data. *Journal of Universal Computer Science*, 16(13):1694–1719, 2010.

[Petrelli *et al.*, 2000] Daniela Petrelli, Elena Not, Carlo Strapparava, Oliviero Stock, et Massimo Zancanaro. Modeling context is like taking pictures. In *CHI'2000 Workshop on Context Awareness*, 2000.

[Pirolli, 1999] Peter Pirolli. *Handbook of Applied Cognition*, chapter 15, Cognitive Engineering Models and Cognitive Architectures in Human-Computer Interaction. John Wiley and Sons, 1999.

[Pixley, 2000] T. Pixley. Document Object Model (DOM) Level 2 Events Specification. W3C Recommendation, November 2000. *http://www.w3.org/TR/DOM-Level-2-Events/*.

[Pohl, 2010] Klaus Pohl. *Requirements Engineering Fundamentals, Principles, and Techniques*. Springer, 2010.

[Presto, 2009] Presto. *Presto Library*. Presto, 2.6.1 edition, 2009.

[Raptis *et al.*, 2005] Dimitrios Raptis, Nikolaos Tselios, et Nikolaos Avouris. Context-based design of mobile applications for museums: a survey of existing practices. In *MobileHCI '05 Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, pages 153–160, 2005.

[Repenning et Ioannidou, 2006] Alexander Repenning et Andri Ioannidou. What makes end-user development tick? 13 design guidelines. In *End User Development*, volume 9 of *Human-Computer Interaction Series*, pages 51–85. Springer Netherlands, 2006.

[RSS, 2009] RSS. RSS 2.0 Specification, version 2.0.11. *http://www.rssboard.org/rss-specification*, March 2009.

[Russell et Norvig, 2009] Stuart Russell et Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[Shoham, 1993] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51 – 92, 1993.

[Singh, 2008] H. Singh. IBM Mashup Center and the InfoSphere MashupHub, Part 1: Get started with InfoSphere MashupHub. *IBM Developer Works*, page 20, 2008.

[Sjøberg *et al.*, 2008] Dag I. K. Sjøberg, Tore Dybå, Bente C. D. Anda, et Jo E. Hannay. *Guide to Advanced Empirical Software Engineering*, chapter Building Theories in Software Engineering, pages 312–336. Springer, 2008.

[Sommerville, 2006] Ian Sommerville. *Software Engineering 8*. Addison Wesley, 2006. ISBN:978-0321313799.

[Sommerville, 2007] Ian Sommerville. *Software Engineering 8*. Addison Wesley, 2007.

[Sun Microsystems, 2007] Inc Sun Microsystems. *Sun Java SystemPortal Server 7.1 Deployment Planning Guide*. Sun Microsystems,Inc, 2007. *http://dlc.sun.com/pdf/819-5073/819-5073.pdf*.

[Thompson, 2008] Rich Thompson. Web Services for Remote Portlets Specification v2.0. OASIS Standard, 2008. *http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html*.

[Tognazzini, 1992] Bruce Tognazzini. *TOG on Interfaces*. Addison-Wesley Professional, 1992.

[TomTom, 2007] TomTom. Total navigation: Our route to the future. anual report and accounts 2007. *http://ar2007.tomtom.com/pdf/tomtom_Ar07.pdf*, 2007.

[Traudt et Konary, 2005] E. Traudt et A. Konary. Software as a Service Taxonomy and Research Guide. Technical report, IDC.com, 2005.

[Ullmann, 1972] Stephen Ullmann. *Semantics: An Introduction to the Science of Meaning*. Oxford, 1972.

[Uschold et Jasper, 1999] Mike Uschold et Robert Jasper. A Framework for Understanding and Classifying Ontology Applications. In *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden*, 1999.

[Valica, 2007] Ecaterina Valica. Mashups - Work You Don't Have To Do . http://www.slideshare.net/valicac/mashups-87355/1, 2007.

[van der Aalst et ter Hofstede, 2005] W.M.P. van der Aalst et A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[van der Aalst, 2009] W.M.P. van der Aalst. TomTom for Business Process Management (TomTom4BPM). In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering (CAiSE'09)*, number 5565 in LNCS, pages 2–5. Springer-Verlag, Berlin, 2009.

[van der Aalst, 2010] W.M.P. van der Aalst. Challenges in Business Process Mining. Technical report, Eindhoven University of Technology, 2010.

[van Dijk, 1997] Teun A. van Dijk. Cognitive context models and discourse. *Language structure, discourse and the access to consciousness*, pages 189–226, 1997.

[van Welie et Trætteberg, 2000] Martijn van Welie et Hallvard Trætteberg. Interaction patterns in user interfaces. In *Proc. Seventh Pattern Languages of Programs Conference: PLoP 2000*, pages 13–16, 2000.

[Vernon *et al.*, 2007] David Vernon, Giorgio Metta, et Giulio Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE Transactions on Evolutionary Computation*, 11(2):151 – 180, 2007.

[Vogels, 2003] Werner Vogels. Web Services Are Not Distributed Objects. *IEEE Internet Computing*, 7(6):59–66, 2003. *http://iat.ubalt.edu/courses/old/idia618.185_Sp04/reading/computer.org.w6059.pdf*.

[Weinberger, 2007] David Weinberger. *Everything Is Miscellaneous: The Power of the New Digital Disorder*. Times Books, 2007.

[Weske, 2007] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures* . Springer-Verlag Berlin Heidelberg, 2007.

[Wicks *et al.*, 2009] Gary Wicks, Egide Van Aerschot, Omar Badreddin, Knut Kubein, Kevin Lo, et Daphne Steele. *Powering SOA Solutions with IMS*. ibm.com/redbooks, 2009. *http://www.redbooks.ibm.com/redbooks/pdfs/sg247662.pdf*.

[Widom et Ceri, 1996] J. Widom et S. Ceri. *Active Database Systems: triggers and Rules For Advanced Database Processing*. Morgan Kaufmann Publishers, 1996.

[Yahoo, 2009] Yahoo. *Konfabulator Reference Manual*. Yahoo, 4.5 edition, 2009. *http://manual.widgets.yahoo.com/*.

[Yu *et al.*, 2007] Jin Yu, Boualem Benatallah, Regis Saint-Paul, Fabio Casati, Florian Daniel, et Maristella Matera. A framework for rapid integration of presentation components. *WWW '07: Proceedings of the 16th international conference on World Wide Web*, May 2007.

[Zittrain, 2008] Jonathan Zittrain. *The Future of the Internet And How to Stop It*. Yale University Press New Haven and London, 2008.

le cnam

Emilian PASCALAU

Vers un développement Web orienté utilisateur
Towards Web User-Centric Development

le cnam

Résumé :
World Wide Web (WWW) est devenu le plus grand dépôt d'informations que l'homme ait jamais assemblé et il est en croissance continue. WWW s'est transformé en un environnement génératif qui favorise l'innovation par le développement des technologies et par un changement dans la perception des gens sur le Web et comment l'utilisent. Le nouveau WWW ou l'Internet de l'Avenir est celui d'un Internet des Services et un Internet des Objets.
Naturellement, une série des questions se posent à partir de ce contexte : comment filtrez-vous les choses pour créer plus de valeur que vous obtenez actuellement ? Comment pouvez-vous regrouper les choses d'une manière intelligente et facile au lieu de la faire dans votre tête? Le monde ne peut pas être décrit sans ambiguïté, alors comment pouvez-vous permettre aux utilisateurs de traiter avec le monde à leur manière, en fonction de leur compréhension? Levine dans son livre "Cluetrain manifesto" a argumenté que les marchés sont conversations, alors comment peut-on impliquer les utilisateurs dans la conversation ? Comment les utilisateurs peuvent être autorisés à la consommation facile des services, de l'information, des choses qu'ils trouvent autour?
Cependant, la conception et le déploiement d'un tel logiciel capable d'interaction directe et l'autonomisation de l'utilisateur final reste toujours un problème. On a, d'une part, les utilisateurs qui ont des idées, mais qui n'ont pas l'environnement technique et les capacités en programmation pour faire eux-mêmes le développement. D'autre part, on a un grand volume des données, ressources et services qui qui pourraient être regroupées à la fois en termes de données, mais le plus important, en termes de comportement d'innover et de créer nouveaux objets.
Notre objectif dans cette thèse est de combler ce manque d'outils qui sont capables d'une interaction directe et l'autonomisation des utilisateurs finaux, de manière unifiée. Ainsi, notre principale contribution dans cette thèse est le développement d'une approche holistique pour les systèmes basés sur le Web qui sont centrés sur l'utilisateur et qui intègrent des données, les services et le comportement disponible sur le Web 2.0.
Mots clés :
développement web orienté utilisateur, mashups, services web, web 2.0, contexte, comportement

Abstract :
World Wide Web (WWW) has become the greatest repository of information that man has ever assembled and it is continuously growing. WWW transformed itself into a generative environment that fosters innovation through the advance of technologies and a shift in people's perception of the Web and how they use it. The new WWW or Future Internet is that of an Internet of Services and Internet of Things.
Naturally, a series of questions arise from this context: how do you filter things to create more value than you currently get? how do you aggregate things in an intelligent and easy way instead of doing it in your head? The world cannot be described unambiguously, so how can you allow users to deal with the world in their own way, based on their understanding? Levine in his book "Cluetrain manifesto" was arguing that markets are conversations so how can users be involved in the conversation? How can users be empowered with easy consumption of the services, information, things that they found around?
However design and deployment of such software capable of direct interaction and empowerment of the end-user is still an issue. We have on one side users that have ideas, but do not have technical background and lack programming skills to do the development by themselves. On the other side, we have large amounts of data, resources and services that could be aggregated both in terms of data, but most important in terms of behavior to innovate and create new things.
Our goal in this thesis is to address this lack of tools that are capable of direct interaction and empowerment of end-users, in a unified manner. Thus our main contribution in this thesis is the development of a holistic approach for web based systems that are user-centric and that integrate data, services and behavior available on the Web 2.0.
Keywords :
web oriented end-user development, mashups, web services, web 2.0, context, behavior