JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Markus Weißenbek, BSc

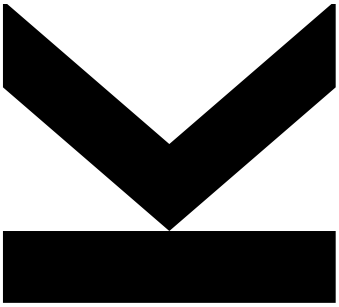
Submitted at
Institute of Telecooperation - Department of Cooperative Information Systems

Supervisor
Assoc.Prof. Mag. Dr. Wieland Schwinger, MSc

Co-Supervisor
a.Univ.-Prof. Mag. Dr. Werner Retschitzegger

July 2019

Towards decentralized Volunteer Management Systems – Conceptual Approach & Architecture



Master Thesis
to obtain the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

SWORN DECLARATION

I hereby declare under oath that the submitted Master's Thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited.
The submitted document here present is identical to the electronically submitted text document.

Linz, July, 10 2019

ABSTRACT

Volunteering is an invaluable contribution to our society, playing a tremendous role not only in overcoming large scale crises as shown by the refugee crisis in 2015, but also in much smaller scale activities like mentoring children or teenagers. Volunteering almost always involves cooperative activities involving other people and volunteers and is generally an unpaid activity aiming to help society. While a lot of people are volunteering for large, globally acting organizations like the Red Cross, also the number of volunteers, volunteering for other individuals outside of any organizational structure should not be understated. Volunteering typically is associated with health care and ambulance service, volunteering goes far beyond those domains and also spans across educational services, social services, environmental protection or disaster relief. Not least due to the sheer amount of volunteers, extensive IT-support for volunteer-involving organizations (VIO) is indispensable leading to the establishment of volunteer management systems (VMS). VMS are not only used to efficiently allocate volunteers to volunteering opportunities, but try to support all phases of the volunteering process including volunteer acquisition, volunteer coordination and volunteer retainment, e.g. by incorporating diverse motivation strategies.
While existing VMS already excel at supporting VIOs with, e.g., the management of tasks or resources, the support for volunteers often lacks behind, disregarding the utilization of already earned competences for other VIOs and external parties, e.g., recruitment agencies, due to the lack of missing interoperability. The system iVolunteer described in this thesis tries to tackle these problems, by centering its functionality around volunteers and investigating "how their engagement can be digitized and exploited in a life-long way". Thus, based on a systematic identification of requirements, a digital volunteer ecosystem is proposed, focusing on incorporating various VIOs, providing interoperability between them and allowing volunteers to exploit their competences not only within these VIOs but also beyond them. Mitigating the current trend of data silos, volunteers are given sovereignty over their earned competences, providing them with means to privately store and manage them, leading to the necessity to provide for appropriate verification thereof, incorporating a blockchain in order to establish trust.

KURZFASSUNG

Freiwilligenarbeit stellt einen unbezahlbaren Beitrag für unsere Gesellschaft dar und spielt eine enorme Rolle, nicht nur bei der Überwindung großer Krisen, wie die Flüchtlingskrise im Jahr 2015 gezeigt hat, sondern auch bei weitaus kleineren Aktivitäten wie der Betreuung von Kindern oder Jugendlichen. Freiwilligenarbeit umfasst meist kooperative Aktivitäten, an denen andere Menschen bzw. Freiwillige beteiligt sind und ist im Allgemeinen eine unbezahlte Aktivität zum Wohle der Gesellschaft. Während sich viele Menschen freiwillig für große, global agierende Organisationen wie das Rote Kreuz engagieren, sollte die Anzahl jener Freiwilligen, die sich für andere Personen außerhalb einer Organisationsstruktur engagieren, nicht unterschätzt werden. Freiwilligenarbeit wird in der Regel mit Gesundheits- und Krankenpflegediensten in Verbindung gebracht, allerdings geht sie weit über diese Bereiche hinaus und erstreckt sich auch über Bildungseinrichtungen, soziale Dienste, Umweltschutz oder Katastrophenhilfe. Nicht zuletzt aufgrund der großen Zahl von Freiwilligen ist eine umfassende IT-Unterstützung für Freiwilligenorganisationen (VIO) durch sog. Freiwilligenmanagementsystemen (VMS) unabdingbar. VMS werden nicht nur verwendet, um Freiwilligen effizient Freiwilligentätigkeiten zuzuweisen, sondern versuchen, alle Phasen des Freiwilligenprozesses zu unterstützen, einschließlich der Freiwilligenakquise, des Freiwilligenkoordination und der Freiwilligenbindung, z.B. durch Einbeziehung verschiedener Motivationsstrategien.
Während bestehende VMS bereits umfangreiche Funktionalität bei der Unterstützung von VIOs bieten, z.B. bei der Verwaltung von Aufgaben oder Ressourcen, ist die Unterstützung für Freiwillige selbst häufig nicht vorhanden und bspw. vernachlässigt die automatisierte Erfassung und Nutzung bereits erworbener Kompetenzen für andere VIOs und externe Parteien, z.B. Personalagenturen. In dieser Masterarbeit wird das System iVolunteer vorgestellt, welches versucht, diese Probleme zu lösen, indem dezidierte Funktionalitäten für Freiwillige zur Verfügung gestellt werden und untersucht, ”wie ihr Engagement digitalisiert und lebenslang genutzt werden kann”. Auf der Grundlage einer systematischen Ermittlung der Anforderungen wird daher ein digitales Freiwilligen-Ökosystem vorgeschlagen, das sich darauf fokussiert, verschiedene VIOs einzubeziehen, für Interoperabilität zwischen ihnen zu sorgen und es Freiwilligen zu ermöglichen, ihre Kompetenzen nicht nur innerhalb dieser VIOs, sondern auch außerhalb dieser zu nutzen. Um den aktuellen Trend von Datensilos entgegenzuwirken, erhalten Freiwillige die Souveränität über ihre erbrachten Leistungen und erworbenen Kompetenzen und bekommen die Möglichkeit, sie privat zu speichern und zu verwalten. Dies macht eine Verifizierung dieser erforderlich, die auf Basis sog. "Blockchain-Technologie" erfolgt um deren Nachweisbarkeit sicherzustellen und somit entsprechendes Vertrauen zu schaffen.

ACKNOWLEDGMENTS

At this point, I would like to thank all those who have significantly supported me throughout the realization of this Master's thesis. Most of all, I would like to thank both of my supervisors, a.Univ-Prof. Mag. Dr. Werner Retschitzegger and Assoc. Prof. Mag. Dr. Schwinger MSc for their extremely professional and competent care. For their constructive criticism, wealth of ideas and the endless effort, which you have spent over the last years. For my parents Monika and Johann as well as their respective significant others Alois and Hermine, unwaveringly supporting me throughout my life, without them I would not have come this far. I'm tremendously happy abount having won the first price in the lottery of birth.
For my best friends - Markus, Amer, Daniel, Ines, Norbert, Andreas, Dustin and Florian - for their mutual respect and acceptance of who I am, enabling me to continuously strive for becoming a better person.
Thank you Philipp Starzer and Berthold Roiser for jointly implementing the proof-of-concept prototype. It was a great cooperation, resulting in an incredible prototype representing the groundwork of this thesis.
Last of all, thank you, Alexander, for initially leading me down the rabbit-hole into the phenomenal world of knowledge and learning, showing me its playfulness, thereby igniting my flame to learn each and everything.

PREFACE

This thesis originates from the project „iVolunteer - Eine Digitale Plattform zur Nutzbarmachung informeller Kompetenzen im Freiwilligenbereich“, which is funded by the Austrian Research Promotion Agency (FFG, ProjectNr. 871494) under the COIN-program line.
The implementation of iVolunteer was realized in cooperation with my student collegues Philipp Starzer and Berthold Roiser. In my thesis, I describe the conceptual part of iVolunteer used as groundwork for the implementation, while their theses focus on certain implementation aspects of iVolunteer.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

Contents

In this chapter, an introduction to volunteering in general and consequently to volunteer management systems with their related challenges are presented and discussed. Afterwards, the research projects *iVolunteer* as well as its predecessor project *Cooperative Activities* are presented, representing two different approaches of volunteer management systems (VMS). Finally, the problem definition is presented followed by a short outline of this thesis.

1.1 VOLUNTEERING

Volunteering is an integral part of our society in various different domains - health care, education, environmental protection or disaster relief [1]. While according to Merrill [2] the understanding of the term *volunteering* can vary depending on the country and culture, it is still always characterized by the following four features. (i) Volunteering involves active participation or contribution in the form of time, energy or competence and does not include just the contribution of financial or material resources, which would rather fall into the domain of donations or sponsoring. (ii) Volunteering is uncoerced, i.e., volunteers contribute their time, energies and competences freely without compulsion. (iii) Even though volunteers sometimes are compensated for their personal and material expenses, volunteering is never primarily motivated by any financial gain. Last, (iv) the outcome of volunteering focuses on the common good as opposed to individual enrichment.
Since volunteers scarcely work the number of hours normal workers do, the number of volunteers is typically measured in *full-time equivalent* (FTE) workers [3] in order to adequately compare the volunteer workforce. The global volunteer workforce in 2018 accounted for 109 million volunteers [4] measured in FTE workers, exceeding for example the workforce of the Russian Federation or Japan. According to the *State of the World's Volunteerism Report* by the *United Nations Volunteers* (UNV) [4] around 30% of the 109 million FTE volunteers participate *formally* through an organization (e.g., Red Cross) and 70% *informally* for other individuals. For Europe, the report states, that of the 29 million FTE volunteers, 26.7% are *informal* FTE workers and 73.3% *formal* ones.

Interestingly, a similar statistic for Europe by Salamon et al. [1] suggests otherwise with a distribution of 59% FTE *formal* volunteers and 31% *informal* volunteers. The remaining 10 % are attributed to cooperatives and social enterprises. A study by the Johns Hopkins University [5] investigated 13 countries, including Canada, Israel, France, USA, Japan and Norway, and found, that on average, volunteers make up more than 7 % of the total workforce for all of these countries as well as that for 6 of the 13 countries the average workforce accounts for more than 10 %. In Europe, the volunteering sector comprises volunteers equal to almost 30 million FTE workers [1] and is therefore barely behind the manufacturing (32 million) and trade (30.7 million) sector [1].
Furthermore, not all volunteers act similarly in their way of engaging. Kapsammer et al. [6] show a broad spectrum of volunteers starting with (i) *Patchwork Volunteers*, engaged in various different VIOs throughout their life, to (ii) *Engagement Hoppers*, which spontaneously getting active depending on their own availability and demand for volunteers and finally, (iii) *Crowd Volunteers* focusing primarily on online micro-tasks (e.g., helping and presenting solutions on *stackoverflow*[1]). Irrespective of the way volunteers engage in, volunteering is additionally one of the best places where informal learning takes place, i.e., learning outside of traditional educational institutions like schools or universities, usually resulting in the attainment of new competences, which are not only valuable for other VIOs, but can also be used in the labor and education market. In order to manage not only the voluntary work to optimize processes from an economic point of view, but also exploit the potential of volunteers, the need for adequate IT-support arose, leading to a plethora of *volunteer management systems*.

1.2 CHALLENGES OF VOLUNTEER MANAGEMENT SYSTEMS

A volunteer management system brings together volunteers and volunteering opportunities and allow for the scheduling, allocation and execution of volunteering tasks and projects. It further provides means for communication and coordination in order to encourage collaboration [7]. Due to the vast amount of volunteers and VIOs, already a lot of VMSs emerged [8], often focusing on different goals and supporting different phases of the volunteering process (e.g., volunteer acquisition, management or encouragement, etc.) [7]. Unlike the proposed VMS of this thesis, most VMS favor a VIO-centric approach, trying to support VIOs as best as possible by managing their volunteers and tasks with the drawback, that volunteers rarely receive certification about tasks they carried out respectively their earned competences. However, if a VMS does allow volunteers to receive certifications, they rarely have full access over their data since they are stored centrally at the VIO, similar to data silos. This raises the problem, that while volunteers perform a tremendous amount of work, they get awarded nearly nothing that shows their commitment and efforts and due to the missing

1 www.stackoverflow.com

interoperability of existing VMSs, their gained competences can rarely be exploited for purposes outside of the VIO, e.g., for job applications. While the challenge of allowing volunteers to better exploit their competences they earned already represents a major challenge, VMS suffer from much more problems as Kapsammer et al. [7] found:

INHOMOGENEOUS CONGLOMERATE OF RESOURCES AND TASKS As already stated, VMSs try to support VIOs among other things with the allocation of tasks, involving bringing together tasks and available resources, which are needed in order to carry out the task. Both, available resources and resources a task needs are often highly inhomogeneous and therefore an allocation between them is far from trivial. For *human resources*, i.e., volunteers, the challenge is the inhomogeneity between their competences, interests resp. personalities and the needed competences and qualifications required to carry out tasks. Additionally to human resources, also *non-human resources* need to be considered during the task allocation.

CONFIGURATION OF TASKS Due to different internal processes, the configuration of tasks wrt. their *structural* and *behavioural relationships* have to be considered. Structural relationships typically incorporated the splitting of tasks into several sub-tasks, thereby building a hierarchical task structure, thus reducing complexity and allowing to manage tasks better by distributing sub-tasks to various volunteers. Regarding behavioral relationships, appropriate mechanisms allowing for the utilization of temporal dependencies between tasks have to be contemplated, e.g., where a specific task has to be finished before another can start.

FLEXIBLE ALLOCATION ALLOWING BROKERAGE AND NEGOTIATION Since voluntary work is rarely paid, keeping volunteers motivated becomes a necessity in order to inspire volunteers for certain tasks and to achieve a sustainable level of commitment of them over time. Thus, the matching between a volunteer's competences as well as interests and tasks plays a key role, especially because exact matching might not suffice. Thus, a flexible kind of brokerage employing different *matching strategies*, which not only focus on the matching but also incorporating some sort of well-balanced effort distribution may be beneficial. Further expanding on this idea, instead of either carrying out a task or not, a *negotiation* between VIOs and volunteers could establish a more finely-grained way of defining the contribution of a volunteer to a task.

ADAPTATION- AND MOTIVATION-ORIENTED ASSESSMENT In order to identify potential improvements of the outcome of voluntary tasks, appropriate assessment mechanisms along the volunteering process are crucial. These mechanisms should not only allow an assessment by volunteers or the VIO, but also by the beneficiaries of the voluntary work. Moreover, the assessment should comprise different parts of the volunteering, including the task and its outcome, volunteers themselves (e.g., engagement, etc.), allocation of tasks, social aspects or gain of expertise.

CONTINUOUS EVOLUTION The last challenge is to incorporate evolution mechanisms into a VMS, thus, enabling the improvement of the voluntary process itself by the allowing adaptation of tasks (e.g., extending, merging and decomposing of tasks) and resources (e.g., incorporating achievements).

1.3 RESEARCH PROJECT – "COOPERATIVE ACTIVITIES (CRAC)"

The challenges of VMS described in the previous section where targeted by the pre-decessor project of iVolunteer *Cooperative Activities* (CrAc), which was also funded by the Austrian Research Promotion Agency (FFG; ProjectNr. 845947) under the COIN-program line. The aim of CrAc was the development of interdisciplinary concepts allowing for dynamic, profile-based assignments of cooperative tasks to volunteers. CrAc focused on continuously adapting tasks and participants to real conditions by incorporating evaluation criteria and aims at the following three goals [9]. First, the support of *dynamic, multi-granular profiles* for volunteers as well as for tasks, supported by information extraction and ontology learning to structure and fill up the profiles. Second, intelligent techniques for the matching of tasks to volunteers by regarding similarity measures and spatio-temporal calculations of the profiles as well as by incorporating balanced task allocation mechanisms. Last, CrAc employed crowd-based assessment and evolution mechanisms leading to a dynamization of the profiles and the matching techniques. All of the three goals should result in an increase in the effectiveness of volunteering and stimulate creative potential within volunteer organizations.

1.4 RESEARCH PROJECT – "IVOLUNTEER"

In contrast to its predecessor project *CrAc*, *iVolunteer* is centered around volunteers with a clear focus on how the engagement of volunteers can be digitized and exploited in a life-long way and proposes a digital platform for utilizing formal and informal competences, which are acquired through voluntary engagement (see Fig. 1.1). iVolunteer strives for the following three goals. The first goal is, that volunteers should be able to oversee and track their volunteering engagement (i.e., their so-called ”volunteer footprint”), which will be stored in an individual "*digital volunteer pass*". It should not only consist of the activities that were carried out but also list competences a volunteer earned, respectively the feedback they received. The volunteer footprint should allow for the decentralized storage at the sole disposal of the volunteer, thus giving the volunteers more sovereignty and responsibility for their data and therefore directly counteracts the current trend to store data in centralized data silos. These competences are then used to support further competence acquisitions and applications at *iVolunteer Marketplaces*, which represent the second goal of iVolunteer. These *iVolunteer*

Marketplaces bring together volunteers and volunteering opportunities by coordinating offers and needs between VIOs and volunteers. iVolunteer marketplaces are generally independent of any certain VIO, thus allowing also informal volunteering to take place by incorporating informal volunteers and their help-seeking people directly. Nevertheless, VIOs should also have the possibility to participate by operating one or more marketplaces. Therefore, iVolunteer supports both, formal volunteering for an organization and informal volunteering directly for other persons. The third goal of *iVolunteer* is to integrate *volunteer encouragement* by supporting gamification mechanisms in order to lower entry barriers into volunteering and maximize life-long engagement.

Figure 1.1: iVolunteer Overview

1.5 PROBLEM DEFINITION

This thesis is situated in the realm of the iVolunteer project, contributing mainly to the first two research goals of iVolunteer, the *iVolunteer Footprint* and the *iVolunteer Marketplaces*. In particular, a conceptual approach and a corresponding system architecture tackling both of them should be developed and parts of them also implemented by means of a proof-of-concept prototype. This concept should generally allow for the management of decentralized *iVolunteer Marketplaces*, on which volunteering opportunities and volunteers should be coordinated. Volunteers should be able to participate in these marketplaces and thereby developing their volunteer footprint, which should be stored decentralized at the sole disposal of the volunteer. Furthermore, volunteers should be able to use their volunteer footprint across multiple marketplaces. These three aspects which are in the focus of this thesis - decentralized MPs (i.e., VIO-independent) , decentralized footprint storage (i.e., counteracting Data Silos) and decentralized footprint exploitation (i.e., cross-MP and beyond) - characterize our notion of a "Decentralized VMS" being in the center of iVolunteer. The implementation of the proof-of-concept prototype, which has been realized together with my colleagues Philipp Starzer and Berthold Roiser is described in more detail in their thesis

works. In this thesis only a glimpse of the implementation is given, while focusing especially on the conceptual view of the proposed VMS.

1.6 THESIS OUTLINE

Based on the described vision of iVolunteer and the problem definition given, this thesis is structured as follows. Chapter 2 introduces the *requirements* of iVolunteer, which further expand on the brief problem definition given above. The requirements represent the foundation of the *conceptual approach* in Chapter 3 and the *system architecture* in Chapter 4. In Chapter 5, a comparison of iVolunteer with existing VMSs is conducted, showing the areas where typical VMSs have drawbacks and where iVolunteer shines. Finally, in Chapter 6 the *conclusion* respectively the *future work* are discussed.

2 REQUIREMENTS

Contents

In this section, the emerged user roles are presented, followed by a description of the requirements for each core component of iVolunteer (see Fig. 2.1) - (i) the *marketplace component*, (ii) the *cross-marketplace component* and (iii) the *footprint component*. Within iVolunteer, each formal resp. informal volunteering organization should operate one dedicated *marketplace component*, serving as their very own VMS and representing the organization's marketplace, which coordinates between demand and supply of voluntary work and performing the bonding between help-seekers and volunteers. The *cross-marketplace component* serves as mediator between the volunteers and the marketplace components, allowing for their initial registration as well as constituting as central communication point between users and the other two components, i.e., the *marketplace components* and the *footprint component*. Last, the *footprint component* manages the volunteering footprint by giving volunteers full sovereignty while still guaranteeing, that it, nevertheless, can be verified for correctness (i.e., whether the entries of the volunteer footprint can be trusted to be true).

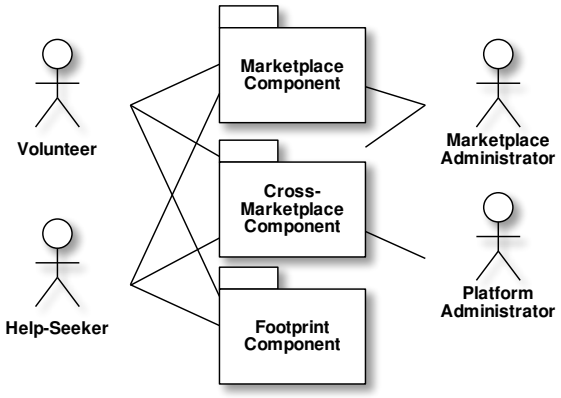


Figure 2.1: Use Case Packages of iVolunteer

2.1 USER ROLES

Within iVolunteer, four user roles emerged, representing the different actors for the use cases presented in the following sections. The user roles within iVolunteer are the (i) *volunteer*, who carries out tasks managed by (ii) *help-seekers* on the marketplaces. Each marketplace component of iVolunteer is managed by (iii) *marketplace administrators*, while the cross-marketplace component is operated by the (iv) *platform administrator*. In addition to user roles, a permission system should be implemented enabling a fine-grained configuration of which user is able to perform which functionality.

2.1.1 Volunteer

Within iVolunteer, *volunteers* play the central role by offering not only their human resources (i.e., time or competences), but also additional non-human resources (e.g., tools, vehicles) in order to help others by performing tasks mediated on certain marketplaces (see Sect. 2.2). In order to participate in a marketplace, an according subscription is necessary, before being able to carry out tasks as described in Sect. 2.2.2 or synchronizing their volunteer footprint with the marketplace (see Sect. 2.4.1).

2.1.2 Help-Seeker

Help-seekers are the counter-part to volunteers within iVolunteer. While volunteers carry out voluntary tasks, help-seekers are responsible for creating and managing them on the *marketplace component*. Similar to volunteers, help-seekers must be able to subscribe to marketplaces in order to operate on them. Furthermore, help-seekers must be able to create tasks and manage them (e.g., assign volunteers to tasks, start and finish tasks; see Sect. 2.2). Since the management of tasks can be critical and involve disclosed information about participants, organizations might want to control, whether help-seekers can freely participate in

their marketplaces or not. Especially for formal volunteering, VIOs might want to refrain help-seekers outside of the organization to join. Therefore, iVolunteer should support a mechanism to allow or disallow help-seekers to join, by setting adequate *subscription rules*, which are further explained in Sect. 2.2.1.2.

2.1.3 Marketplace Administrator

The *marketplace administrator* is responsible for setting up, managing and customizing a *marketplace component*. The user who initially registers a new *marketplace component* at the *cross-marketplace component* will automatically become its marketplace administrator. Besides choosing the name and description of the marketplace, marketplace administrators have three other configuration possibilities. First, they must be able to set *subscription rules* (see Sect. 2.2.1.2) for users, stating whether users are allowed to subscribe to a marketplace as volunteer or help-seeker. Second, marketplace administrators should be able to define available task life-cycles (see Sect. 2.2.2.1), handling how tasks are processed by defining the steps a task goes through the user interactions of help-seekers and volunteers that are responsible for the different steps of the task life-cycle. After the marketplace administrator configured the desired set of task life-cycles, help-seekers can create tasks attributed with one of these task life-cycles. Third, marketplace administrators should be able to configure the set of competences (see Sect. 2.2.3) that are needed at the respective marketplace. This configuration possibility is not least valuable since organizations often not only utilize general-purpose competences (e.g., teamwork), but also employ domain-specific (e.g., Extinguish Fire) or organization-specific ones (e.g., radio course of the Fire Brigade).

2.1.4 Platform Administrator

The *platform administrator* is responsible for managing the *cross-marketplace component* and its infrastructure (see Sect. 4.1.1). The platform administrator should be able to register new *marketplaces* at the *cross-marketplace component*, thereby publishing it and enabling users to subscribe to them as volunteers and help-seekers. Furthermore, the platform administrator should be able to update and delete already existing marketplaces from the cross-marketplace component.

2.2 MARKETPLACE COMPONENT

The *marketplace component* should offer the possibility to coordinate between supply and demand for volunteering by employing a highly configurable *task management*, handling volunteering tasks and the allocation of volunteers to tasks. In order to find appropriate tasks for volunteers and vice versa, a *recommendation management* is needed, recommending tasks based on the similarity between a volunteer's competences and requirements of tasks as well as by considering other volunteer or task-specific preferences like time or priority. In order to

handle competences, a *competence management* is needed, covering both, formal and informal competences as well as employing competence derivation mechanisms in order for volunteers to receive these competences based on certain task fulfillments. To further complement tasks, a *resource management* should allow for the management of human and non-human resources within a VIO, enabling users to claim resources for the timespan of tasks. Furthermore, as previously discussed, marketplaces should be able to be configurable leading to the *marketplace configuration management*.

2.2.1 Marketplace Configuration Management

The *marketplace configuration management* is responsible for allowing marketplace administrators to configure their marketplace component. In general, the configuration possibilities for marketplaces could be numerous and would therefore go far beyond the scope of this thesis. For this thesis, the marketplace configuration management has two main responsibilities - (i) the management of the *marketplace formality type* and (ii) the handling of *subscription rules*.

2.2.1.1 *Marketplace Formality Type*

As already mentioned, there has to be made a distinction between formal and informal volunteering, directly influencing the user structure of a marketplace and the functionality they are able to use. The main difference between the two types of volunteering is, that formal volunteering is generally done for organizations, while informal volunteering is done for individuals (e.g., helping a neighbour). Since iVolunteer should support both, formal and informal volunteering, appropriate mechanisms are needed to fulfill the requirements for both types of volunteering. In essence, the difference is, that on marketplaces operated by VIOs - called formal marketplaces - help-seekers are generally employees of the respective VIO, while on informal marketplaces (i.e., for informal volunteering), anyone should be able to be a help-seeker. Therefore, appropriate mechanisms (i.e., subscription rules) are required to restrict the registration of help-seekers on formal marketplaces, while allowing all users to subscribe to an informal marketplace as both, volunteer and help-seeker.

2.2.1.2 *Subscription Rules*

In order to not only allow users to subscribe to a marketplace as volunteer or help-seeker based on the marketplace formality type (i.e., formal or informal marketplace), but to enable marketplace administrators to adapt the preconditions to subscribe to a marketplace in a more fine-grained way, *subscription rules* should be introduced. These subscription rules should not only restrict access for help-seekers, but also for volunteers allowing a marketplace to be publicly accessible or private for volunteers. For example, the subscription rule could state, that help-seekers can only subscribe to a marketplace after they received an invitation e-mail.

2.2.2 Task Management

The task management (see Fig. 2.2) is responsible for supporting ways to manage and carry out volunteering tasks. Since VIOs have their very own processes of handling volunteering tasks, a highly configurable task management needs to be integrated into the *marketplace component* allowing them to realize and integrate them within iVolunteer. In the following sections, requirements regarding task management are presented, which are inspired by Mundbrod and Reichert [10], [11], resp. [12], starting with the *task configuration*, i.e, how tasks can be adapted to the needs of VIOs, followed up by the incorporation of *task templates* in order to simplify the task creation process and finishing with the functionality to *generate tasks and task templates* from existing tasks and templates in order to simplify the process of creating tasks and task templates.

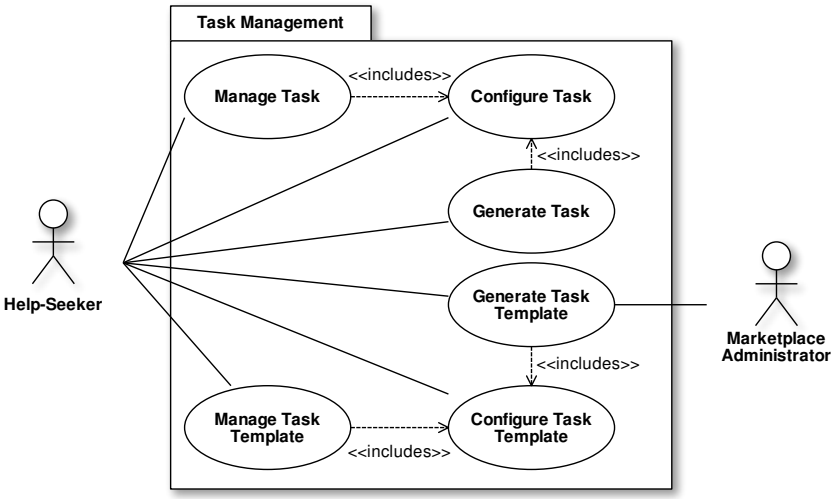


Figure 2.2: Task Management Use Cases

2.2.2.1 *Task Configuration*

Two dimensions of task configuration can be distinguished - (i) intra vs. inter and (ii) structural vs. behavioural (see Fig. 2.3). The intra-configuration comprises all configuration possibilities of a single task, while the inter-configuration covers configuration possibilities of a set of tasks. A structural configuration includes the change of the structure of one task or the structure of a set of tasks, while a behavioural configuration means a change of the execution of a single task or a set of tasks.

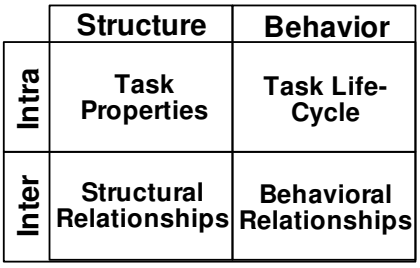


Figure 2.3: Task Configuration Overview

Because of these two dimensions, four possible variations of task configurations emerge:

- **Task Properties**
  It should be possible, that tasks can be adaptable wrt. their internal structure, i.e., their properties. There should be a basic set of properties available for each task, e.g., *name*, *description*, *start time* or *end time*. Furthermore, it should be possible to add additional properties to tasks in order to expand on the basic set of properties. An example of additional properties would be, whether a driver's license is required or not. In order to make these properties as adjustable as possible, not only the name of properties should be customizable, but also its respective value type, e.g., *boolean*, *floating point number* or *character string*. Task properties should be used as a basis to search for tasks, respectively for the matching between volunteers and tasks.

- **Task Life-Cycle Configuration**
  The main requirement for the intra-task configuration besides task properties is to manage how tasks are processed and executed, called configuration of the *task life-cycle* [10]. Task life-cycles are enforced by *task workflows*, controlling the states a task runs through and evaluating, which actions are available at a certain state in the task life-cycle. A task workflow is a graph consisting of states and transitions, depicting the states a task can reach and the transitions in-between. A set of pre-defined states and transitions should be available, including *create task*, *publish task*, *assign task* and *finish task*. Furthermore, there should be a possibility to define new custom states and transitions in order to support the different volunteering task processes of VIOs. A task life-cycle should also be able to incorporate the sequential and parallel execution of these pre-defined and user-defined states.

- **Structural Task Relationships**
  iVolunteer should incorporate *structural task relationships*, revolving around combining tasks together and forming compositions of tasks. In general, hierarchical structures are desirable, since they can depict task compositions with parent-child relationships. Hierarchically structured tasks form a tree, where the root-task is often times the overarching project the tasks belong to, intermediate nodes are grouping tasks, and leaves are executable tasks.

- **Behavioral Task Relationships**
  Last, behavioral task relationship should be incorporated, relating to modelling behavioral interdependencies between tasks. For example, one task has to be performed before another can start (i.e., temporal relationship), which can be necessary due to information dependencies (the result of one task is the prerequisite of another) [13].

2.2.2.2 *Task Templates*

Task templates serve as templates for tasks, in order to quickly create recurring tasks. Similar to tasks, task templates also comprise intra-task configuration possibilities, i.e., (i) task properties and (ii) life-cycle configuration, but do not have any inter-task configuration possibilities, because they are neither structurally nor behaviourally dependent upon other task templates or tasks.

2.2.2.3 *Task & Task Template Generation*

It should not only be possible to generate tasks from task templates, but rather both, tasks and task templates, should be generable from existing tasks as well as from task templates. In both cases, the existing *intra-task configuration*, i.e., task properties and task life-cycle configuration should be taken over to the generated task respectively task template.

2.2.3 Competence Management

As already stated in Chapter 1, volunteering is regarded as a unique opportunity for informal learning, thereby supporting the acquisition of new competences, respectively the refinement of already acquired ones. Thus, not only formal competences acquired through education and training courses, but also informal ones acquired through fulfilling volunteering tasks and working in a volunteering team should be integrated into iVolunteer. It has to be emphasized, that competences within iVolunteer should not only be usable at one specific marketplace, but rather volunteers should be able to transfer the competences from one marketplace to other ones and utilize them there as well. This requirement is further described in Sect. 2.4. The competence management (see Fig. 2.4) of the marketplace component should implement the following use-cases:

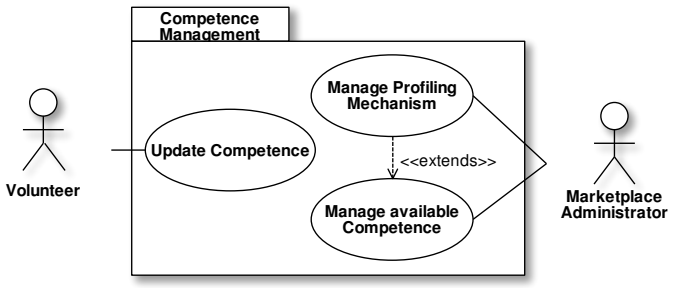


Figure 2.4: Competence Management Use Cases

- **Manage available competences**
  The *marketplace component* should come with a pre-defined set of competences used within iVolunteer, which should be based on already existing competence models and ontologies. A *marketplace administrator* should be able to adapt these existing competences (e.g., adapt possible proficiency levels) as well as create new ones. This is especially necessary because there exist many domain-specific or organization-specific competences, which VIOs should be able to incorporate into iVolunteer.

- **Manage Profiling Mechanism**
  The *marketplace administrator* should be able to change the profiling mechanism, deciding how competences are awarded, updated or taken away. The profiling mechanism should be adaptable from simple rules with pre-conditions, which have to be fulfilled in order to receive the competence, to fully-fledged machine-learning algorithms, which learned how these competences should be awarded. This mechanism should not only be available for formal competences, but also for informal ones, for which the definition of the pre-conditions is much harder to grasp. For example, which qualification is enough to decide whether a volunteer receives the competence "team player".

- **Update Competences**
  Generally, there should be two ways volunteers can update their competences, i.e., receive new competences, change existing competences (e.g., reaching a higher proficiency level) as well as lose already acquired ones. An example where volunteers could lose competences is in the domain of paramedics of the Red Cross, requiring them to have to recertify their defibrillator usage each year.
  The first way, how volunteers should be able to update their competences is, that they should be able to develop them based on their performance of volunteering tasks as well as on interactions within iVolunteer (i.e., with other volunteers and help-seekers). The second way to update the competences is to get credited for already existing competences acquired from other organizations like schools, jobs or other VIOs. Therefore, adequate mechanisms are needed for volunteers to be able to obtain these competences within iVolunteer by showing ample qualification certificates from these other organizations, e.g., a driver's license. The update of competences should be done automatically by the respective profiling mechanism, deciding whether volunteers receive or lose competences.

2.2.4 Resource Management

Another major requirement of the marketplace component closely coupled with task management is *resource management* (see Fig. 2.5), allowing for the management of resources contributed by the VIO and its volunteers. These resources can be claimed and used during the execution of tasks. Resources can be distinguished into two types -

human and non-human resources - both of which should be usable within iVolunteer. Human resources generally refer to the time and competences volunteers contribute, while non-human resources can be any objects brought in by the VIO or the volunteers, which can be utilized throughout the execution of volunteer tasks. Examples for non-human resources are chairs or benches, which can be provided for a certain task. The resource management contains the following use-cases:

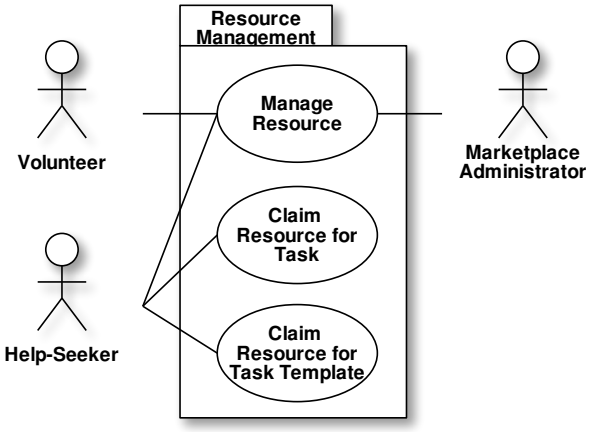


Figure 2.5: Resource Management Use Cases

- **Manage Resource**
  It should be possible to add human and non-human resources to the marketplace component. Each resource should contain at least properties describing *name*, *type* (i.e., human or non-human resource) and *owner*. Furthermore, concepts regarding the availability of resources should be considered, especially because non-human resources often can only be used by one task at the same time. After a resource was created within iVolunteer, the owner of the resource should be able to change or remove it.

- **Claim Resource for Task**
  After resources are initially added, it should be possible to claim them for certain tasks and their defined time period. It should not be possible to claim a resource twice at the same time, except when the resource is explicitly defined to be used concurrently by several tasks (e.g., software).

- **Claim Resource for Task Template**
  Furthermore, it should be possible to append resources to task templates. This process should be identical to the appending of a resource to a task, except that the availability of the resources is not needed to be considered, because task templates have no defined period of execution. The availability of the resource has to be checked, when new tasks are generated from task templates (see Sect. 2.2.2.3).

2.2.5 Recommendation Management

In order to achieve better coordination between demand and supply of voluntary work, a recommendation mechanism (see Fig. 2.6) primarily on the basis of a volunteer's competences and a task's requirements is required. Additionally, further circumstances like the volunteer's temporal and spatial constraints wrt. the task should be taken into consideration. This would allow for a better fit between tasks and volunteers and therefore may increase a volunteer's motivation and contribution. There should be two ways to receive task recommendations - (i) recommendations from the matching algorithm of iVolunteer and (ii) task suggestions from other users within iVolunteer.

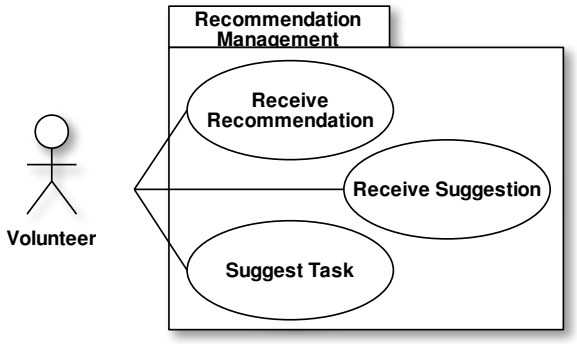


Figure 2.6: Recommendation Management Use Cases

- **Receive Recommendation**
  The first use-case of *recommendation management* revolves around volunteers receiving task recommendation based on their competences and preferences matched against the requirements of tasks. In order to find appropriate fits between tasks and volunteers, matching algorithms of recommender systems should be employed, additionally incorporating interests of volunteers, as well as temporal and spatial information of both, volunteers and tasks.

- **Receive Suggestion**
  In addition to receiving computed recommendations from the iVolunteer platform, it should be possible for volunteers to receive suggestions from other users, i.e., other volunteers or help-seekers. These suggestions could range from last-minute requests from help-seekers, because they quickly need another volunteer for a certain task to friends suggesting tasks to each other in order to carry them out together.

- **Suggest Task**
  In order for volunteers to receive suggestions, it must be possible for users to suggest tasks to them in the first place.

2.3 CROSS-MARKETPLACE COMPONENT

The *cross-marketplace component* should represent the central communication point between users and the other two components, i.e., marketplace component as well as the *footprint component*. The first main requirement of the cross-marketplace component is to handle the different types of users described in Sect. 2.1, called the *user management* (see Sect. 2.3.1).The second requirement is to establish *multitenancy* (Sect. 2.3.2) by enabling the registration of new and the management of existing marketplace components, thereby allowing interoperability of the marketplaces under one common platform while still ensuring, that the data of each VIO (i.e., through separate marketplace databases) is separately stored from each other. The third major requirement revolves around managing the *social management* (Sect. 2.3.3) of iVolunteer, thereby allowing users to communicate with each other across the boundaries of the marketplaces. Finally, the last requirement of the cross-marketplace component is to allow users to configure their dashboard, which is described by the *GUI management* (Sect. 2.3.4).

2.3.1 User Management

Since iVolunteer should manage personal data of users, especially of volunteers and help-seekers, means for authentication need to be in place to protect unauthorized access to the data. Therefore, all users - volunteers, help-seekers, marketplace administrators and platform administrators - are required to be logged in before they should be able to use it. Thus, an appropriate registration and authentication mechanism has to be provided. Since iVolunteer is a distributed application, not only the authentication at the cross-marketplace component has to be managed, but also the authentication at the marketplaces. Furthermore, since it would be laborious for users to register respectively authenticate at the cross-marketplace component and all their marketplaces separately, one common *single sign-in authentication* should be implemented, which authenticates users at both, the cross-marketplace component and their marketplaces using only a single registration, respectively login. This should minimize the effort for users, subscribed to several marketplaces at once. To assure, that access to the cross-marketplace component and the marketplace components is authenticated, a secure token technology with validation date should be applied, ensuring a platform wide single sign-in authentication.

2.3.2 Multitenancy Management

iVolunteer should support multiple formal and informal volunteering organizations independently of each other by allowing them to register and manage their very own marketplace at the cross-marketplace component. In order for volunteers and help-seekers to use the marketplace of a particular organization, they need to subscribe to it and thereby express their wish to contribute to that organization. The requirement of multitenancy (see Fig. 2.7) is formulated in the following features:

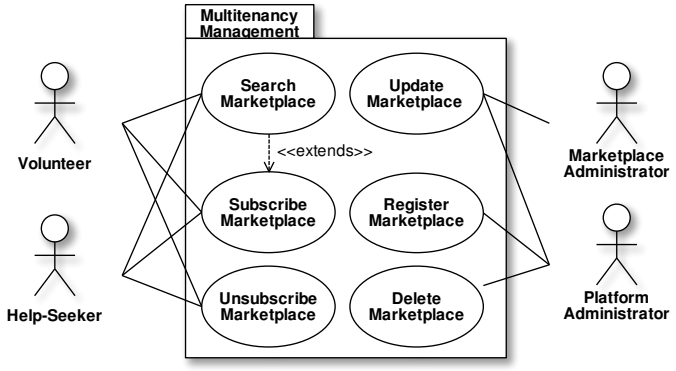


Figure 2.7: Multitenancy Management Use Cases

- **Register Marketplace**
  In order to be able to use a marketplace within iVolunteer, an organization has to register their marketplace at the cross-marketplace component, thereby publishing the marketplace to volunteers and help-seekers enabling them to subscribe and contribute by carrying out tasks for the respective organization.

- **Update Marketplace**
  Once a marketplace is registered at the cross-marketplace component, it should be possible to change the details of the marketplace like name or location.

- **Delete Marketplace**
  It should be possible to delete registered marketplaces and thereby remove them from iVolunteer, making it unavailable for volunteers and help-seekers to join it anymore. All volunteers and help-seekers, that already joined the marketplace beforehand will no longer be able to operate on the marketplace through the *cross-marketplace component*.

- **Search for Marketplace**
  Users should be able to search for registered marketplaces by entering keywords, which should be matched against the name of the marketplace, its respective organization operating the marketplace, the domain of the organization or properties of tasks published at the marketplace.

- **Subscribe to Marketplace**
  Once a marketplace is registered at the cross-marketplace component, volunteers and help-seekers should be able to subscribe to them as long as the respective *subscription rules* allow it (see Sect. 2.2.1.2).

- **Unsubscribe from Marketplace**
  After users joined a marketplace, they, of course, should be allowed to leave it again, thereby deleting their data (e.g., published competencies) from the respective marketplace. After users have unsubscribed, they should no longer be able to access the marketplace (e.g., search and register for tasks) without subscribing to

it again. Nevertheless, earned achievements (e.g., competences) must still be usable if they were synchronized before by utilizing the mechanisms of the footprint component (see Sect. 2.4).

2.3.3 Social Management

Additionally to multitenancy and thus following for the management of marketplace components, the cross-marketplace component is also responsible for *social management* within iVolunteer. Social management should provide opportunities to have social relationships of different users across multiple marketplaces and therefore represents an essential cornerstone of iVolunteer since these social relationships are one of the main reasons, why volunteers are actively engaged [14] and is thus of utmost importance in order to keep them motivated. Social management is divided into the following categories - (i) *social relationship management*, (ii) *social awareness management* and (iii) *social communication management* - each of them forming an essential requirements for iVolunteer.

2.3.3.1 *Social Relationship Management*

Social relationship management should consist of mechanisms for modelling and integrating social relationships between users (i.e., other volunteers or help-seekers). The social relationship management should at least contain a dedicated *friend system* as well as an integration of a *group system*. The friend system serves as the stepping stone for the realization of further social interactions (cf. below). In general, the friend system should either support bi-directional relationships, where both users have each other listed as a friend or uni-directional relationships, where one user can follow another user, without having to follow them as well. Additionally to friends, groups enable users to build a community together with other users. Groups should be configurable to either be private or public, meaning that the membership to the group is either restricted or not.

2.3.3.2 *Social Awareness Management*

Social awareness management should be responsible for giving users information (i.e., making them aware) about upcoming events or activities of friends. In order to achieve awareness, an activity feed in the form of a timeline should be incorporated into iVolunteer. The timeline gives users a chronological history of their friend's activities, posts, upcoming events, earned achievements as well as information related to their volunteering tasks, like when a task was finished. Each entry of the timeline should allow for comments in order to further social intercourse.

2.3.3.3 *Social Communication Management*

The *social communication management* should offer two different communication possibilities among users - *user-chat* and the *group-chat*. While

the user-chat allows only for the bi-directional chat between two users, the group-chat broadcasts the messages to all participants of the group.

2.3.4 GUI Management

The GUI management supports users of iVolunteer with additional configuration possibilities, when it comes to their user interface. While these configuration possibilities could be numerous, like the selection of different fonts, colors, location and size of user interface elements (UI elements), this thesis should only focus on the configuration of the dashboard, thereby enabling users to create their own user-defined dashboards with their preferred UI elements. Users should be able to select them from a predefined set of UI elements and freely place them in their preferred location. Furthermore, users should be able to create multiple dashboards with different UI element placements.

2.4 FOOTPRINT COMPONENT

The *footprint component* should be responsible for allowing volunteers and marketplaces to utilize the volunteer footprint, which should record activities carried out by a volunteer throughout their volunteering life in terms of an individual *digital volunteering pass* [6]. Examples of what a volunteer footprint can comprise are (i) all tasks a volunteer carried out, (ii) all formal and informal competences they acquired, (iii) their feedback as well as feedback they receive (e.g., feedback regarding tasks, projects, etc.) and (iv) social interactions they have with other volunteers or help-seekers. Typically in VMSs, volunteer footprints are stored centrally at large data centers leading to huge data silos and users, which often don't know what information is stored about them. To counteract this trend of data centralization and privacy infringements, iVolunteer should store the volunteer footprints decentralized for the exclusive disposal of the volunteer in the so-called *local repository*. Since volunteers are empowered by this decentralization and could therefore alter their volunteer footprint at will, appropriate verification mechanisms are required, allowing for the checking of whether entries of the volunteer footprint are indeed correct. The footprint component is therefore divided into two packages - (i) the *footprint verification management*, responsible for verifying footprint entries and (ii) the *footprint synchronization management*, covering the synchronization mechanisms between marketplaces and volunteers.

2.4.1 Footprint Verification Management

The *footprint verification management* should provide mechanisms to verify volunteer footprint entries and check whether they are valid or not. Due to the empowerment of volunteers by having full leverage over their data, adequate verification methods are needed in order to detect, whether footprint entries provided by volunteers to a marketplace are valid or not. For example, volunteers could append additional competences in their volunteer footprint, which must be detected as

invalid. Otherwise no marketplace could trust the footprint of volunteers, thus rendering the volunteer footprints useless. The footprint verification management therefore should have two main focal points - (i) the warranty to store each footprint entry at a trusted place besides the local repository of volunteers and (ii) using this second storage of the footprint entries for the verification of footprint entries from the local repository. This second storage of the footprint entries should only be used for the verification of footprint entries and will therefore be called *verification storage* from now on. The verification storage has to fulfill the following two characteristics - (i) *trustability*, (ii) *immutability* - and has to enforce the (iii) *verifiability* and (iv) *information obfuscation* of footprint entries.

2.4.1.1 *Trustability*

As described earlier, the verification storage should be used for verification of footprint entries, thus establishing trust between volunteers and marketplaces whenever volunteers try to synchronize footprint entries (e.g., acquired competences) to marketplaces. In order to establish this trust, not only the verification mechanism but also the verification storage itself has to be trusted, otherwise the whole verification mechanism could not be trusted at all.

2.4.1.2 *Immutability*

The verification storage must guarantee the immutability of all contained footprint entries, i.e., it should not be possible to change an entry once it exists in the verification storage. By enforcing immutability of the footprint entries, the verification mechanism can be assured, that the footprint entries of the verification storage can be trusted to be true and therefore achieves a similar goal than the previously described characteristic *trustability*. The main difference between them is, that immutability guarantees, that existing footprint entries are not changed, while trustability assures that no new footprint entries are added to the verification storage, that are valid.

2.4.1.3 *Information Obfuscation*

Since the volunteer footprint, which contains critical personal information about volunteers is stored in the verification storage, adequate measures to save and protect the data from unauthorized access have to be implemented. But rather than authorizing, which information can be accessed by which user of the verification storage, obfuscation mechanisms should be applied, while still guaranteeing *verifiability* (cf. below). Since the verification storage has to be publicly available in order to guarantee, that anyone can verify footprint entries, information obfuscation secures that the information stored is not maliciously exploitable by anyone except for verification purposes.

2.4.1.4 *Verifiability*

Since the verification storage is mainly used to verify, whether footprint entries from the local repository of volunteers are valid or not, it has

to guarantee, that the verification mechanism can be applied to its entries, even though the previously explained *information obfuscation* characteristic will make the entries unreadable. In order to achieve both, information obfuscation and verifiability, one-way hashing algorithms should be applied to the entries of the verification storage to obfuscate them. In order to verify footprint entries from the local repository, they have to be hashed with the same algorithm and compared with entries from the verification storage. If an entry in the verification storage can be found, the respective footprint entry of the local repository is valid, respectively invalid, if the comparison is without result.

2.4.2 Footprint Synchronization Management

The *footprint synchronization management* (see Fig. 2.8) is responsible for the synchronization of footprint entries between a volunteer's local repository and the marketplaces and incorporates footprint verification from the footprint verification management (see Sect. 2.4.1). In order to clarify the terminology of the different locations, where footprint entries can be synchronized to, the volunteer footprint of the local repository will be called *private volunteer footprint*, while the volunteer footprint a marketplace can use will be called *public volunteer footprint*. Since volunteers can not only be subscribed to several marketplaces at once, but also can share different parts of their private volunteer footprint with each marketplace, there must exist one public volunteer footprint for each marketplace a volunteer is subscribed to. Therefore, footprint entries can exist in three different places - (i) in the local repository, (ii) at a marketplace and (iii) in the verification storage. While the footprint synchronization management only manages the synchronization between the first two, i.e., between the local repository and the marketplaces, it still incorporates the verification mechanisms using the obfuscated footprint entries from the verification storage. Since it is solely up to the volunteers to synchronize between their private and public volunteer footprints, both of them can contain either the same footprint entries, have overlapping entries or have diametrically different footprint entries. In order to support the synchronization between private and public volunteer footprints, three mechanisms must be implemented - (i) *downloading footprint entries* from the public volunteer footprint to the private one, (ii) *publishing footprint entries* from the private volunteer footprint to one public footprint and (iii) *revoking footprint entries* from one public footprint.

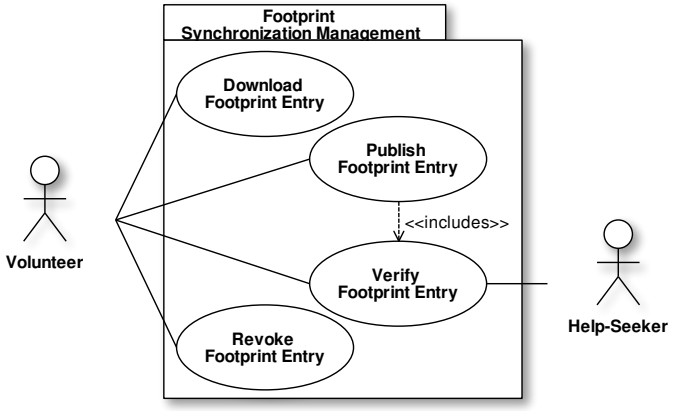


Figure 2.8: Footprint Synchronization Management Use Cases

- **Download Footprint Entry**
  Once a marketplace issues a footprint entry to volunteers, they can synchronize it from the public volunteer footprint into their private volunteer footprint. Once it is synchronized, the volunteer can choose to verify it, respectively publish it to other marketplaces (cf. below).

- **Publish Footprint Entry**
  Publishing a footprint entry from the private footprint to a public footprint makes it accessible to the respective marketplace and allows it to use the entries, e.g., utilizing the footprints for task recommendation etc.

- **Revoke Footprint Entry**
  In order to signal a marketplace to no longer utilize a footprint entry, volunteers can revoke it from the public footprint. The footprint entry will then be deleted from the respective public footprint.

3 CONCEPTUAL APPROACH

Contents

Following up on the requirements of the proposed VMS iVolunteer as stated in Chapter 2, an explanation of the conceptual approach will be given. For each main component stated in the last chapter, one or more packages will be derived and discussed in the following sections (see Fig. 3.1).

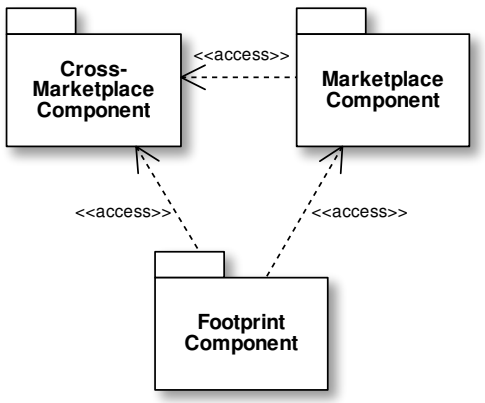


Figure 3.1: Conceptual Package Structure

3.1 CROSS-MARKETPLACE COMPONENT

The responsibilities of the cross-marketplace component were already briefly explained and it is accordingly divided into four packages as stated in Sect. 2.3 - *user management*, *multitenancy management*, *social management* and *GUI management* which will be described in the following (see Fig. 3.2).

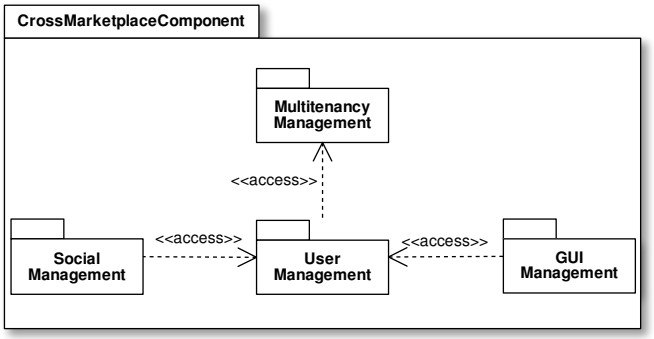


Figure 3.2: Cross-Marketplace Component Package

3.1.1 User Management

The *user management* package (see Fig. 3.3) provides registration and login capabilities for *Users* with different *UserRoles* (i.e., *volunteers*, *help-seekers*, *marketplace administrators* and *platform administrators*; see Sect. 2.1) not only to the *general management platform* (Sect. 2.3) but also to all subscribed *marketplaces* (Sect. 2.2).

3.1.1.1 *User*

The class *User* is identifying a person within iVolunteer and is therefore necessary for authentication purposes including the registration and login into iVolunteer. Each person only needs to register once at the cross-marketplace component with an *email* address and *password*, which form the user's authentication credentials. Furthermore, a user can select a *username*, which will be displayed throughout the system. After the registration is successful, the user is able to log in by using their respective authentication credentials. In the future, it is considered to allow for a range of different kinds of registration types other than e-mail (see Sect. 6.2.3.5) and can vary from anonymous authentication without any usage of username or password, to authentication with a legal document (e.g., driver's license or passport). These different authentication mechanisms would allow for a classification of users into different levels of credibility. For example, anonymous users would not be very credible, in contrast to users registered with a legal document.

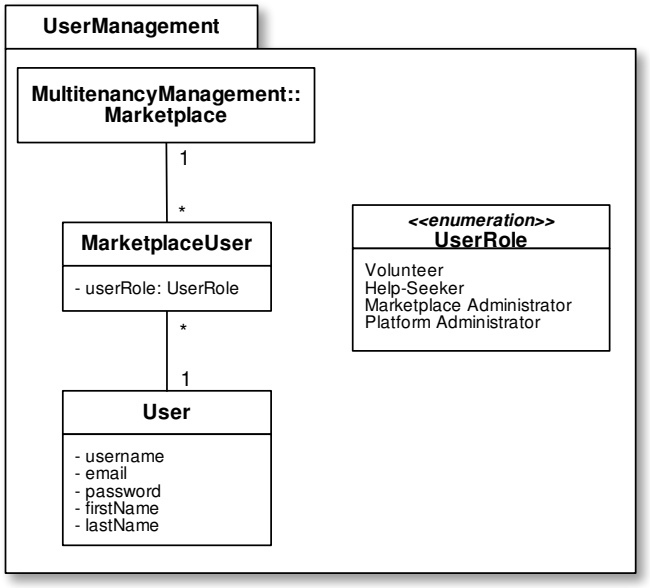


Figure 3.3: User Management Package

3.1.1.2 *UserRole*

Within iVolunteer, users can be attributed four different *UserRoles - volunteers*, *help-seekers*, *marketplace administrators* and *platform administrators*, which were already explained in Sect. 2.1. A user is not limited to only one UserRole, both globally across all marketplaces and within one marketplace, meaning that a user can be volunteer and help-seeker at multiple marketplaces as well as at the same marketplace at the same time. Whether a user can have more roles or not, is solely incumbent upon the subscription rules a marketplace imposes (see Sect. 3.1.2.1). The assignment of which user was allotted which role on one specific marketplace is apparent in the class *MarketplaceUser*.

3.1.2 Multitenancy Management

The *multitenancy management* package (see Fig. 3.4) is responsible for (i) *registration* and (ii) *administration* of *Marketplaces* by their marketplace administrators. The registration of a marketplace publishes the marketplace and makes it available for users to subscribe to as volunteers and help-seekers. The user that registers the marketplace is automatically granted the *UserRole* marketplace administrator and is able to configure the marketplace wrt. (i) its subscription rules (see Sect. 3.1.2.2), which can be set in order to restrict subscription access to the marketplace for both, help-seekers or volunteers and (ii) task workflows in order to configure different types of task life-cycle behaviors (see Sect. 3.2.4.1).

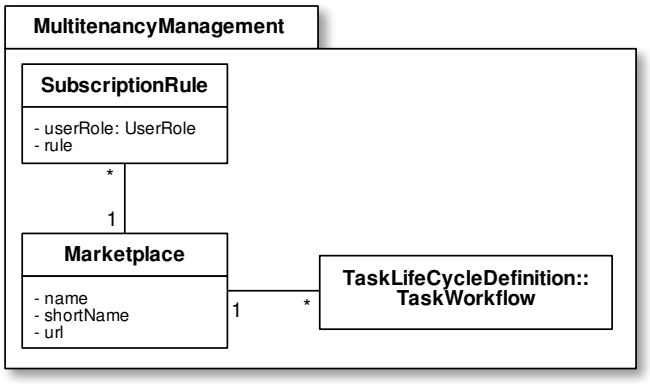


Figure 3.4: Multitenancy Management Package

3.1.2.1 *Marketplace*

The class *Marketplace* is used in order to keep track of registered marketplaces at the cross-marketplace component. Each marketplace consist of a *name*, its *shortName*, as well as the previously mentioned logical address *url*. The *url* is necessary because marketplace components should run in a standalone and distributed way, with the *url* addressing the publicly available address of the marketplace component (see Sect. 4.1).

3.1.2.2 *SubscriptionRule*

SubscriptionRules are used in order to restrict access to a marketplace for certain *userRoles*, i.e., volunteers or help-seekers. If no *SubscriptionRules* are established, both, volunteers and help-seekers can freely subscribe to the respective marketplace. Otherwise, if a *SubscriptionRule* for a certain *userRole* exists, its defined property *rule* has to evaluate to true in order to allow users of the respective *UserRole* access to the marketplace.

3.1.3 Social Management

The *social management* package (see Fig. 3.5) is responsible for handling social intercourse between *Users* within iVolunteer. Since social interactions should not only be limited to marketplaces solely but take place across all marketplaces, the cross-marketplace component is responsible for handling them. Therefore, this package includes a concept for *Groups*, *Postings* and their respective *Comments*. Furthermore, it depicts various *SocialRelationships* between *Users* including the *Friend* and the *FriendOfAFriend* relationship. Last, in order to enable *Users* to "like each other", as well as *Postings*, *Comments* and *Groups*, the interface *Likeable* is introduced. The social management is inspired by already existing social networks such as *Facebook*[1] and *Twitter*[2]. Since the social management is not the core of this thesis, but nevertheless an invaluable part of a VMS, it will only be discussed briefly in the following sections. A much more comprehensive overview of social networks is discussed in [15] and [16].

1 www.facebook.com
2 www.twitter.com

3.1.3.1 *Likeable*

The interface *Likeable* is used in order to enable *Users* the possibility to express their fondness for other *Users*, *Postings*, *Comments* or *Groups*.

3.1.3.2 *Group*

A *Group* is a collection of *Users*, sharing a common interest and interested in communicating with each other. As already stated, *Groups* are not limited to *Users* of one specific marketplace, but can be built up by *Users* across all marketplaces. Furthermore, *Groups* are not only available to volunteers, but all users of different roles (e.g., volunteers, help-seekers, marketplace administrators) can join and form them. Each *Group* consists of a *name*, its *owner* (i,.e. the user, that created the group) and a *members*-property containing a list of all *Users*, who joined the *Group*.

3.1.3.3 *Posting*

A *Posting* is a message of a *User* (=*owner*) addressed to either a *Group* or all *Users* which are in a *SocialRelationship* with the owner of the *Posting* (see Sect. 3.1.3.5). While *Postings* to a *Group* are called *group-postings*, postings to all befriended volunteers are called *friend-postings*. This distinction is determined by the *visibility*-property of a *Posting*, meaning that the visibility is either a certain group or all users in a *SocialRelationship*. The *content* of a *Posting* can vary from a simple text or different types of multimedia files (e.g., video clips, voice recorded message) to shared entities of the iVolunteer application (e.g., shared achievements, tasks, projects, etc.).

3.1.3.4 *Comment*

A *Comment* enables users to state their opinion about an already existing post or another comment. Therefore, *Comments* always refers to either only a *Posting* (= top-level comment) or a *Posting* and another *Comment* (=comment of a comment) and represents a reply (=*content*) from a *User*(=*owner*) to another one regarding the referenced *Posting* or *Comment*. The *Comment* has the same *visibility* as the referenced *Posting*, meaning that *Comments* of a group-posting are also only visible within the same *Group*.

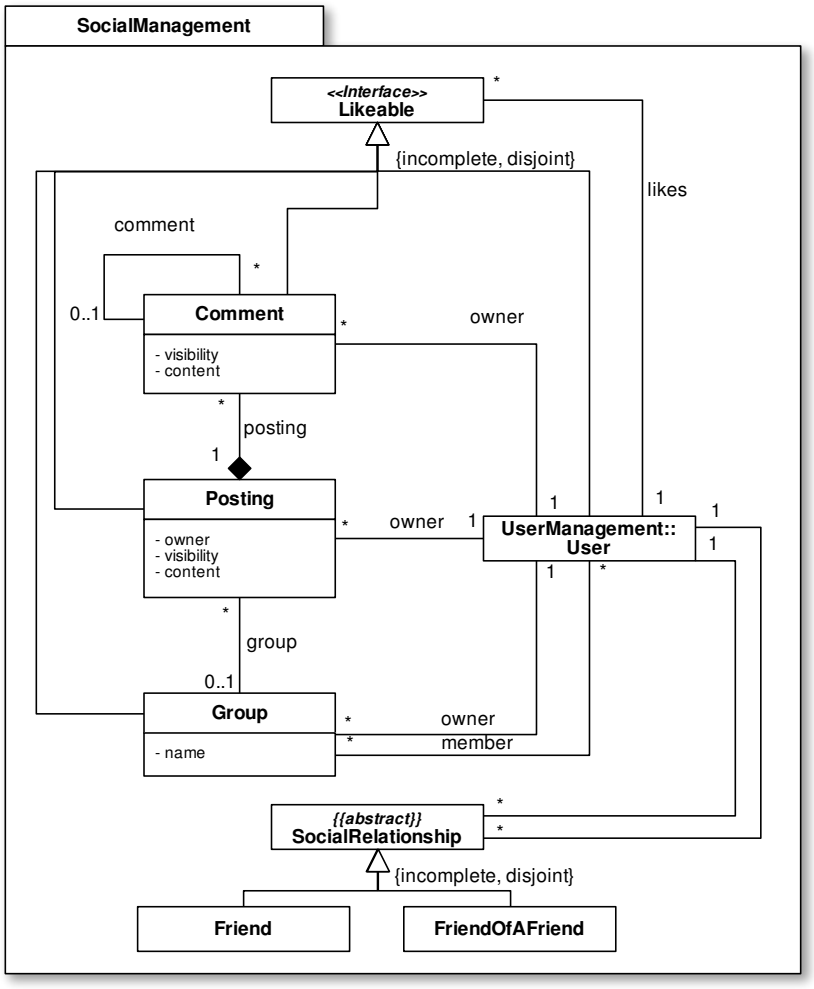


Figure 3.5: Social Management Package

3.1.3.5 *SocialRelationship*

A *SocialRelationship* marks a social connection between two *Users*. While this concept does only include two sub-classes of *SocialRelationship* - *Friend* and *FriendOfAFriend*, in reality, there are numerous such relationships depicted in, e.g., the Social Relationships Ontology[3] (SORON) and discussed in far more detail in [17] which, however, are not yet considered in this thesis.

3.1.4 GUI Management

In addition to the other three packages described so far which provide some core functionality for iVolunteer, the GUI management package (see Fig. 3.6) provides for some cross-cutting functionality regarding the configuration of certain dedicated GUI elements called "*dashboards*" and

3 http://www.bib.uc3m.es/~fcalzada/soron/soron_content/soron

their respective "*dashlets*". A Dashboard is a visual display presenting important information in the form of dashlets.

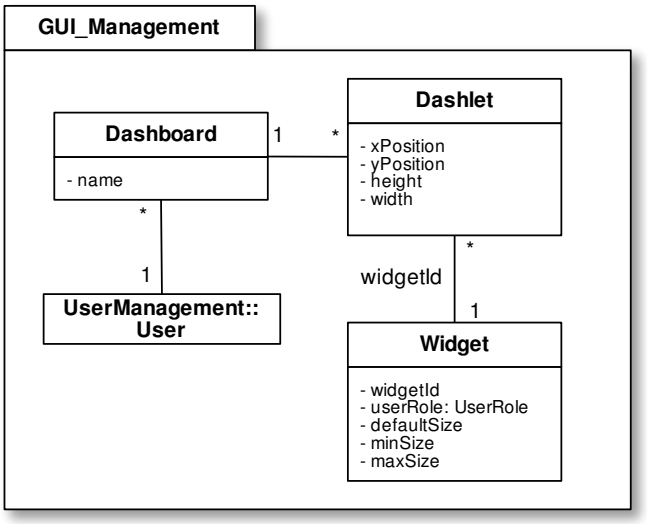


Figure 3.6: GUI Management Package

3.1.4.1 *Widget*

A *Widget* is a view element within iVolunteer and is uniquely identified by the *widgetId*. The *userRole* property marks which *UserRole* is able to use the *Widget*. This is especially necessary because the set of available *Widgets* for volunteers and help-seekers is different. For example, while volunteers have a widget showing all their previously finished *Tasks* in the last month, help-seekers do not have access to this widget. Furthermore, a *Widget* is characterized by a *minSize*, *defaultSize* and *maxSize*, which are used by the *Dashlets* in order to set bounds for the size customization.

3.1.4.2 *Dashlet*

A *Dashlet* wraps a *Widget* and creates a customizable widget, which can be displayed on a *User's Dashboard*. The respective *Widget* is referenced by the *widgetId* and is then placed at the *xPosition* and *yPosition* of the *Dashboard*. The *height* and *width* represent the displayed size of the *Dashlet*. The size of the dashlet has to be between the widget's *minSize* and *maxSize*.

3.1.4.3 *Dashboard*

The dashboard is the first screen displayed to a *User* after a successful login into iVolunteer. *Users* can have more than one *Dashboard*, allowing them to create different views on specific areas of iVolunteer. Each *Dashboard* can be configured to contain a certain set of *Widgets* that can be arranged by the respective *User*. A *Dashboard* comprises a *name* and a list of *Dashlets*.

3.2 MARKETPLACE COMPONENT

The *marketplace component* (see Fig. 3.7) represents the place where volunteers and volunteering opportunities of a VIO come together. Therefore, the marketplace component mainly revolves around the allocation of volunteers to tasks, needing not only a highly configurable task management (i.e., *task structure management* and *task behavior management*), but also further issues, that arise in the broader context of task management, i.e., (i) *management of resources*, organizing human and non-human resources needed for the execution of tasks, (ii) *management of competences*, handling competences needed to carry out tasks, (iii) *management of achievements*, which are awarded after tasks are carried out and (iv) *management of task recommendations* in order for volunteers to receive fitting task suggestions. Furthermore, this package introduces the abstract class *HashableObject*, which lays the groundwork for the synchronization of footprint entries between the local repository (see Sect. 3.3.1) and the marketplace, respectively for the verification (see Sect. 3.3.2.4).

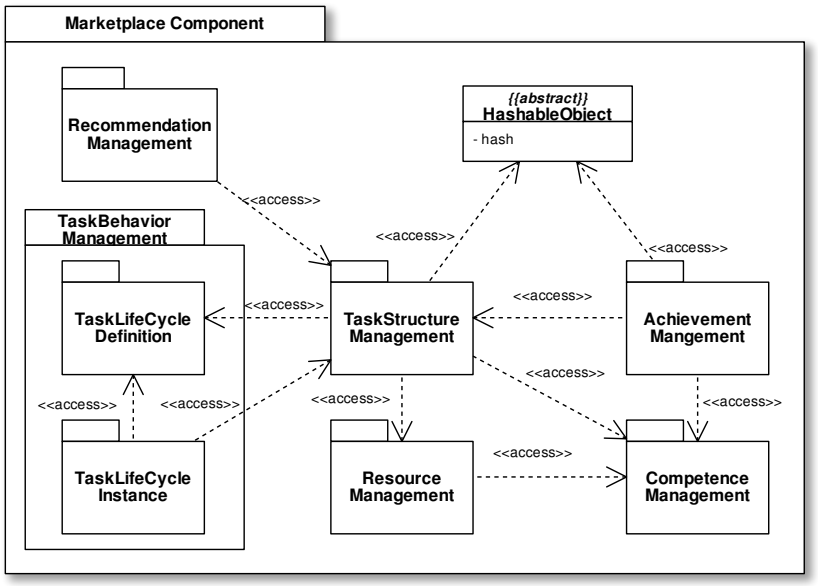


Figure 3.7: Marketplace Component Package

3.2.1 HashableObject

HashableObject is an abstract class (see Fig. 3.7), which enforces for all sub-classes to be hashable, i.e., it must be able to generate a one-way *hash* from the respective objects (see Sect. 3.3.2.2). The class *HashableObject* is generally the superclass of all classes, whose objects have to be stored in the blockchain in order to guarantee verifiability as described in Sect. 2.4.1. Therefore, it represents the superclass of the classes *Achievement*, as well as *Task* and *TaskInteraction*. The class *Achievement* inherits from *HashableObject* because it represents entries of the volunteer footprint stored in the local repository (see Sect. 3.3.1) and thus

have to be verifiable. While the classes *Task* and *TaskInteraction* (see Sect. 3.2.3), are not used for footprint entries, they nevertheless have to be verifiable, because certain *Achievements* are awarded in the context of *Tasks* and thus are linked to them. For example, a volunteer may receive an achievement (e.g., a competence) after a certain task was finished. The achievement is therefore linked to the task and it has to be possible to verify not only the achievement but also the task, that led to the achievement. By not only guaranteeing, that *Achievements* are stored in the blockchain and thus are verifiable, but also storing *Tasks* and *TaskInteractions* in the blockchain, a higher degree of traceability for a volunteer's achievements is reached, i.e., it allows not only for the verification that a volunteer reached a certain achievement, but also how it was reached.

3.2.2 Competence Management

Competences in iVolunteer serve two purposes. First, they are used in order to credit volunteers the competences, they acquire by carrying out volunteering tasks. Second, they are used in order to set preconditions volunteers have to fulfill before they are allowed to carry out tasks. More specific, tasks can be attributed with a set of required competences, which volunteers must have acquired before they are allowed to sign for the task. Clearly, these two purposes are intertwined with each other, because a volunteer earns competences by executing tasks, which can be used to reserve for other tasks, which require these competences. While there exist numerous different taxonomies of competences like by Cheetham and Chivers [18], the taxonomy used in the *competence management* is based on the *competence dimensions* published by Erpenbeck and Sauter [19] because it is a prominent and frequently referenced taxonomy. While the *competence dimension* represent only a stepping stone for the much more comprehensible *competence atlas* [20], the concepts, which involve competences stated in this thesis, i.e., the synchronization into the local repository or the publishing of competences to other marketplaces are independent of the underlying taxonomy of competences and therefore are also applicable for competences structured according to the competence atlas. Therefore, since the core of this thesis does not revolve around the structuring of competences, but rather how volunteers and VIOs can exploit them, the much simpler competence dimensions are used as taxonomy in order to describe the competences.

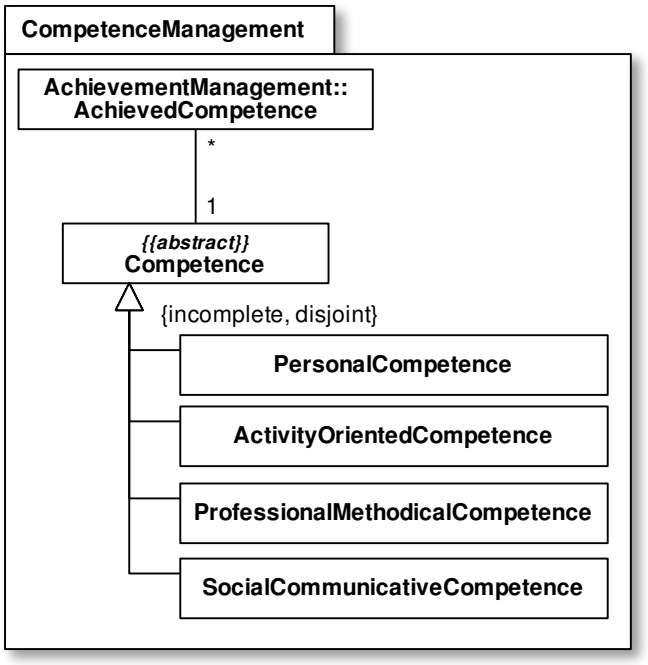


Figure 3.8: Competence Management Package

3.2.2.1 *Competence*

The term *competence* does not have a widely accepted definition [21] and is therefore hard to explain, because competences are such a multi-faceted concept and are either defined as (i) observable performance, (ii) the outcome of one's performance or (iii) underlying attributes of a person [22]. Additionally, there is a difference between the linguistic definition of the terms *competence* and *competency* respectively *competences* and *competencies* [23], which can be explained as follows. A *competence* is described as the capability to carry out an action, while a *competency* is rather the knowledge necessary in order to carry this action out. In order to mitigate confusion about these linguistic differences, from now on only the terms *competence* respectively *competences* will be used throughout this thesis. According to Erpenbeck and Sauter [19], four competence dimensions are commonly used in practice - (i) *personal competences*, (ii) *activity and action-oriented competences*, (iii) *professional-methodical competences* and (iv) *social-communicative competences*. Fig. 3.8 shows the identically-named classes inheriting from the superclass *Competence*.

PERSONAL COMPETENCE

Personal competences comprise skills to be smart and critical towards yourself as well as to develop productive attitudes, values and ideals. Examples for personal competences are loyalty, self-responsibility or credibility. Persons with highly developed personal competences are usually perceived as charismatic and act as role models.

ACTIVITY AND ACTION-ORIENTED COMPETENCE

Activity and action-oriented competences are related to incorporate skills, knowledge, social communication, personal values and implement them into all other competences. Examples are vigor, initiative or execution readiness. Persons with highly developed activity and action-oriented competences are typically competitive, willing to take risks and bear responsibility.

PROFESSIONAL-METHODICAL COMPETENCE

Professional-methodical competences are defined as skills with professional and methodical characteristics used to creatively solve complex problems. Examples are expertise, market knowledge or interdisciplinary knowledge. Persons with highly developed professional-methodical competences are typically task-oriented and reliable as well as characterized by an analytical and methodical way to achieve goals.

SOCIAL-COMMUNICATIVE COMPETENCE

Social-communicative competences involve skills to cooperate and communicate with other people. Examples are the ability to cooperate, as well as adaptability to social norms. Persons with highly developed social-communicative competences typically show a high degree of empathy.

3.2.3 Task Structure Management

The *task structure management* (see Fig. 3.9) combined with the task behavior management (see Sect. 3.2.4 and 3.2.5) form the centerpiece of the marketplace component. The task structure management package is primarily responsible for defining and instantiating *Tasks* and *TaskTemplates*, as well as handling the structural customization of tasks as described in Sect. 2.2.2, i.e., handling user-defined *TaskProperties* and *TaskRelationships* (both, structurally and behaviorally). Whenever a *User* interacts with a *Task* (e.g., a volunteer is assigned to a task), a *TaskInteraction* is created. Furthermore, *Tasks* are integrated into *Projects*, thus grouping them together. Finally, the task structure management combines *Tasks* with their needed input- respectively output-*Resources*.

3.2.3.1 *Task*

The class *Task* represents volunteering tasks, which both, volunteers and help-seekers can interact with. The main information a *Task* consists of is the *name*, *description*, the associated *project*, the *startDateTime* and *endDateTime* of the task. The *taskTemplate*-property references a *TaskTemplate*, a *Task* may be created from. The *inputResources* contains a set of *Resources* (see Sect. 3.2.6.1), which are required in order to carry out the *Task* while the *outputResources* is a set of *Resources* produced from the *Task*. The *properties*-field contains additional *TaskProperties* in order to further structurally customize a *Task*. Furthermore, a *Task* references its *WorkflowInstance* (see Sect. 3.2.5.1), representing the task life-cycle instance for this particular *Task*. The *WorkflowInstance* is an instantiated *TaskWorkflow* and determines the *state* a *Task* is currently in. Examples for *states* are 'CREATED', 'PUBLISHED' or 'FINISHED'.

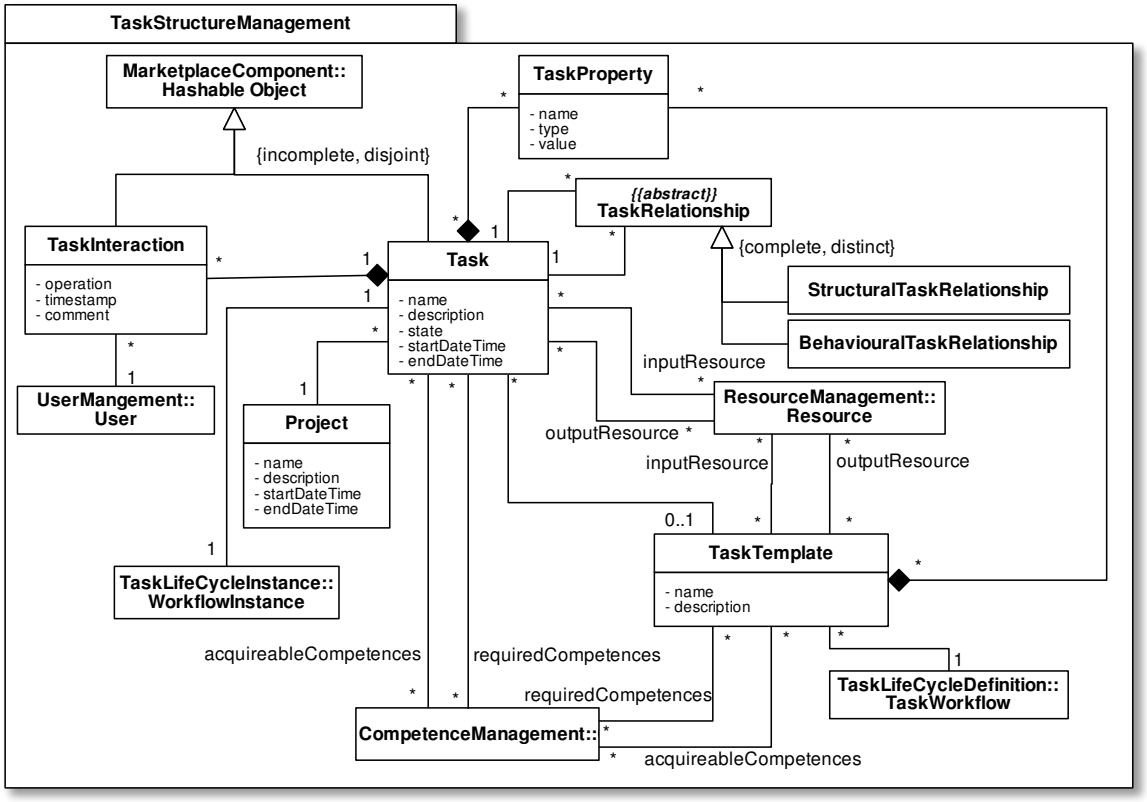


Figure 3.9: Task Structure Management Package

3.2.3.2 *Project*

Seldom, *Tasks* stand alone for themselves, but rather are part of bigger real-world projects. The introduction of *Project* enables help-seekers to group their *Tasks* together, thereby establishing the context of a real-world project. The class *Project* comprises a *name* and a *description* of the project as well as a time-frame delimited by its *startDateTime* and *endDateTime*.

3.2.3.3 *TaskTemplate*

As the name indicates, *TaskTemplates* serve as templates for tasks, in order to quickly create recurring tasks. *TaskTemplates* comprise a *name*, *description*, *inputResources* as well as *outputResources* and additional *properties*, which serve the same purpose as for *Tasks*. Furthermore, a *TaskTemplate* contains a reference to one *TaskWorkflow*, representing a type of a task life-cycle. Whenever a new *Task* is created from a *TaskTemplate*, these properties are taken over and attributed to the respective *Task*, as well as a new *WorkflowInstance* is created from the respective *TaskWorkflow*, i.e., a task life-cycle instance is created from the selected task life-cycle type.

3.2.3.4 *TaskInteraction*

TaskInteractions describe certain interactions between *Users* and *Tasks*, triggered by going from one state of the task's life-cycle to another one. Interactions like reading information about a *Task* do not trigger the creation of a new *TaskInteractions*. Therefore, the *WorkflowInstance*, i.e., the task life-cycle instance is coupled with the class *TaskInteraction*, such that every time the *WorkflowInstance* steps from one *Activity* to the next one, a *TaskInteraction* is created, documenting the result of the life-cycle step. The most typical *TaskInteractions*, which are also part of the default task life-cycle of iVolunteer are (i) task creation, (ii) task publication, (iii) task reservation, (iv) task assignment and (v) task finalization.

3.2.3.5 *TaskRelationship*

The class *TaskRelationship* is introduced in order to represent dependencies between two *Tasks*. As described in Sect. 2.2.2.1, these dependencies can either be structural or behavioural, resulting in the two sub-classes *StructuralTaskRelationship* and *BehaviouralTaskRelationship*.

STRUCTURAL TASK RELATIONSHIP A *StrucuralTaskRelationship* marks a parent-child relationship between two *Tasks* and therefore enables the creation of task hierarchies. *StructuralTaskRelationships* are generally used to split large and complex *Tasks* into smaller ones in order to reduce complexity and make *Tasks* easier for volunteers to carry them out.

BEHAVIOURAL TASK RELATIONSHIP Contrary to *StructuralTaskRelationships*, *BehaviouralTaskRelationships* are used to define behavioural dependencies between tasks. An example of a *BehaviouralTaskRelationships* would be a temporal dependency, where one certain *Task* has to be finished before another one can be carried out.

3.2.3.6 *TaskProperty*

TaskProperties are used to customize the structure of a single *Task* or *TaskTemplate* (see Sect. 2.2.2.1). A *TaskProperty* comprises a *name*, its value *type*, and the *value* itself. Common examples for *types* are boolean, floating point number or string.

3.2.4 Task Life-Cycle Definition

The task life-cycle definition package (see Fig. 3.10) describes the building blocks a task life-cycle can be built from. In general, a task life-cycle represents a unidirectional graph consisting of nodes and edges. The nodes represent actions of the life-cycle while the edges define transitions between them. The following model is heavily inspired from the workflow management framework *Activiti*[4], which was used in the implementation of the proof-of-concept prototype.

4 https://www.activiti.org/

3.2.4.1 *TaskWorkflow*

The class *TaskWorkflow* describes the task life-cycle definition (see Sect. 2.2.2), determining the behavior of the respective task. A *task life-cycle* represents a state-machine containing all states a task can reach as well as all transitions between those states. A task life-cycle can be established by incorporating *life-cycle support into a task management* [10], dealing with the preservation and evolution of tasks. The main difference between task life-cycle and life-cycle support for a task management is that the former deals with the adaptation of a single task, while the latter depicts the process of how task life-cycles are established and evolved. Each *TaskWorkflow* contains a *workflowName* and a *workflowUri*, locating the workflow definition file (e.g., a valid task workflow described on the basis of BPMN [24]). Furthermore, each *TaskWorkflow* can be activated respectively deactivated by setting the *active* flag accordingly, thereby making new tasks available respectively unavailable for life-cycle definition.

3.2.4.2 *FlowElement*

In general, a *TaskWorkflow* consists of a series of *FlowElements*, which are stepped through as the task life-cycle is processed. However, *FlowElements* cannot be strung together arbitrarily, but rather a workflow is built like a graph from the juxtaposition of nodes and edges, which are called *FlowNode* and *SequenceFlow* (cf. the corresponding subclasses below). The abstract class *FlowElement* comprises a *name* and a list of additional *properties*, which can be used to state further information about the *FlowElement*.

3.2.4.3 *SequenceFlow*

A *SequenceFlow* connects two *FlowNodes* with each other, therefore enabling the workflow to step from one *FlowNodes* to another by following the respective path. This path is stored by the *SequenceFlow* in the form of the *source* respectively *target FlowNode*. By storing the *FlowNodes* explicitly as *source* and *target*, a *SequenceFlow* provides therefore always a unidirectional path between them. Furthermore, a *SequenceFlow* can be customized with a *conditionExpression*, managing whether the flow will be followed or not. In general, a *conditionExpression* is only needed, when several paths emanate from a *FlowNode* (see Sect. 3.2.4.10). In this case the *conditionExpression* is evaluated in order to select the path that should be followed. This property is especially required for *Gateways* (see Sect. 3.2.4.10).

3.2.4.4 *FlowNode*

As already stated, a *FlowNode* is a *FlowElement* representing an action in the workflow. A *FlowNode* is further distinguished into either an *Activity* or a *Gateway* (cf. subclasses below). While an *Activity* is an active node, where an action is either started automatically or by a user, a *Gateway* is an intermediate node that handles one-to-many *SequenceFlows* between *FlowNodes*.

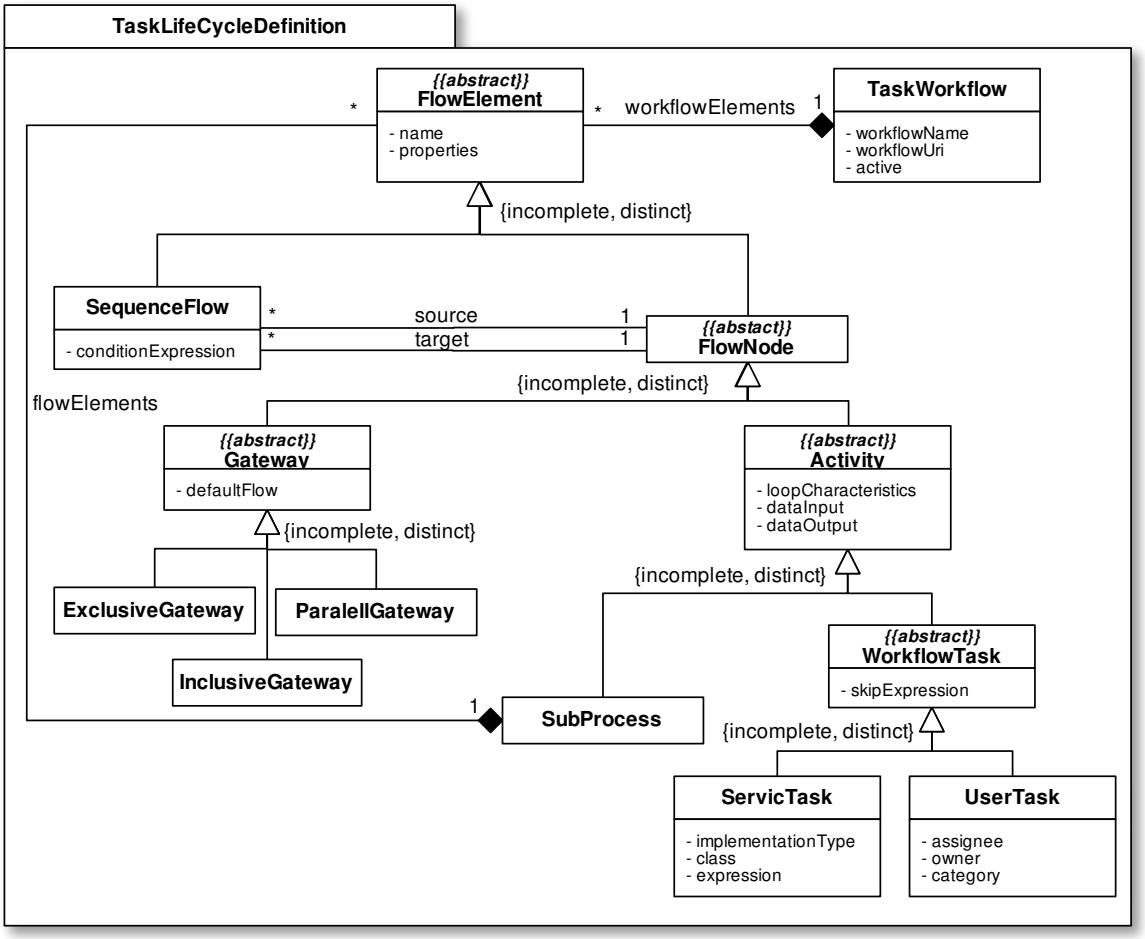


Figure 3.10: Task Life-Cycle Definition Package

3.2.4.5 *Activity*

An *Activity* represents an active state in the workflow, where either a process is started automatically or the workflow is waiting for further user input. Each *Activity* contains properties for *dataInput* and *dataOutput*. The *dataInput* is often times needed for internal computations of the activities, which can result in additional *dataOutput* that can be queried after an *Activity* is processed. The property *loopCharacteristics* is necessary because an *Activity* can not only trigger its internal action once but several times. The *loopCharacteristics* is an expression, that marks how often the activity's computation is triggered. For example, an *Activity* can be triggered a fixed amount of times or until a certain condition evaluates to true.

3.2.4.6 *SubProcess*

SubProcesses are required to structure a workflow into smaller sub-workflows and therefore enabling reuse and reducing complexity.

3.2.4.7 *WorkflowTask*

In contrast to a *SubProcess*, which can be seen as a composition of multiple *Activities*, a *WorkflowTask* represents only a single action. Additionally, a *WorkflowTask* comprises a *skipExpression*, regulating whether to skip the *WorkflowTask* or not depending on the given expression.

3.2.4.8 *UserTask*

A *UserTask* is a special form of *WorkflowTask*, which is executed only after a user interaction is triggered. The *UserTask* stores the *assignee*, i.e., the user that is able to perform the *UserTask*, the *owner* of the *UserTask* and the *category* of the task. The *category* is used to group *UserTasks* together, for example into *default UserTasks* and *special UserTasks*, which can be used to distinguish between different user-interface implementations.

3.2.4.9 *ServiceTask*

In contrast to *UserTasks*, which are only executed after a user interaction is initiated, *ServiceTasks* are triggered automatically, when the workflow reaches the respective *WorkflowTask* and executes their associated implementation. The implementation of a *ServiceTask* can be either defined by a fixed *class* or given by an *expression*. The *implementationType* regulates which of the two types should be applied for the respective *ServiceTask*.

3.2.4.10 *Gateway*

Gateways allow for the introduction of branches, i.e., forks and joins, enabling to model much more complex workflows as compared to single path workflows. *Gateways* are dynamically evaluated and depending on the sub-class - *ExclusiveGateway*, *ParalellGateway* or *InclusiveGateway* - and their respective outgoing *SequenceFlow's conditionExpressions* one or all paths of the *Gateway* are processed (cf. below). The class *Gateway* contains a *defaultFlow*, referencing the *SequenceFlow*, that should be taken if no other expression is given. While a *Gateway* is a single *FlowNode*, in the workflow a gateway is always used for both, the forking and the joining part as depicted in Fig. 3.11. The forking gateway is located at the start of the branch and consists of at least one incoming flow and several outgoing flows. The joining gateway is the counterpart to the forking gateway and is placed at the end of the branch with several incoming flows and only one outgoing flow.

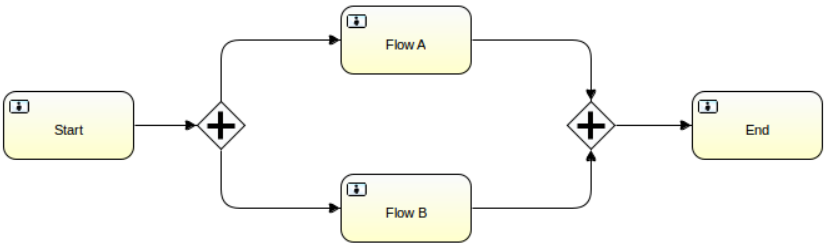


Figure 3.11: Exemplary Gateway

3.2.4.11 *ExclusiveGateway*

ExclusiveGateways are *Gateways*, where exactly one outgoing *SequenceFlow* is processed. For this type of *Gateway*, all outgoing *SequenceFlows* of the forking gateway have to be annotated with a *conditionExpression*, evaluating to either true or false, determining if the path should be processed or not. In general, the *conditionExpression* of all outgoing *SequenceFlows* have to be complete and distinct, meaning that for each possible outcome of the evaluation of all *conditionExpression* at least one (completeness) and at most one (distinctness) *SequenceFlow* must be taken, i.e., exactly one outgoing *SequenceFlow* for each possible outcome of the evaluation of all *conditionExpression*.

3.2.4.12 *ParallelGateway*

In contrast to the *ExclusiveGateway*, the *ParallelGateway* does not have annotated outgoing flows at the forking *Gateway*, because all *SequenceFlows* are taken and processed concurrently. At the joining gateway all *SequenceFlows* are synchronized with each other, thus waiting until each *SequenceFlow* is finished. After all *SequenceFlows* are finished, the workflow proceeds.

3.2.4.13 *InclusiveGateway*

The *InclusiveGateway* is a hybrid gateway resembling to some extent the *ExclusiveGateway* and the *ParallelGateway*. Outgoing *SequenceFlows* of the forking gateway are annotated with *conditionExpressions* like in *ExclusiveGateways*, but more than one paths can be taken (i.e., all *SequenceFlows* whose *conditionExpression* evaluates to true). Therefore, the *conditionExpression* has to only be fulfilled the completeness characteristic. Similar to the *ParalellGateway*, the joining gateway synchronizes all executed flows and then proceeds with the next workflow step.

3.2.5 Task Life-Cycle Instance

While the *task life-cycle definition* package (see Fig. 3.12) is responsible for modelling *TaskWorkflows*, the *task life-cycle instance* package handles the execution of them by creating a *WorkflowInstance* from the *TaskWorkflow*. Once a *TaskWorkflows* is modelled, it can be executed arbitrarily often and concurrently (i.e., several workflow executions of the same defined workflow at the same time) since each *WorkflowInstance* is independent of all others. Within iVolunteer, each *Task* is coupled with one *WorkflowInstance*, which handles its task life-cycle accordingly to the defined *TaskWorkflow*.

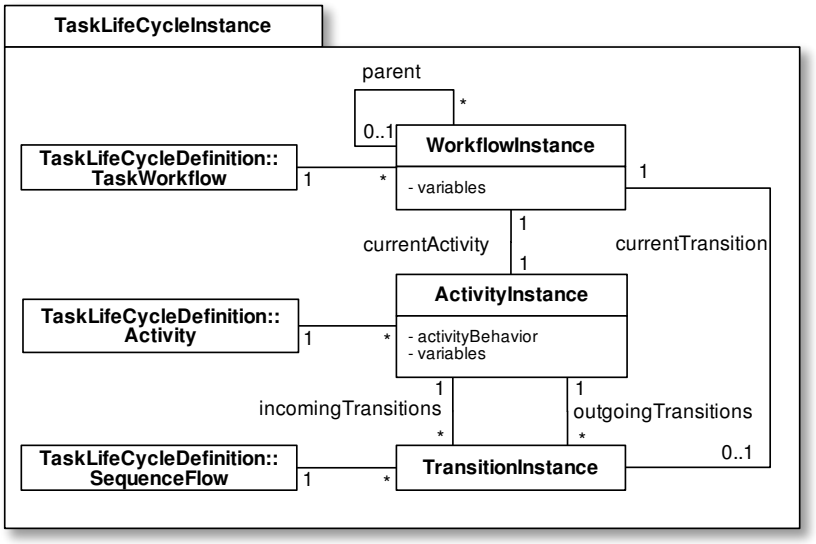


Figure 3.12: Workflow Instance Package

3.2.5.1 *WorkflowInstance*

The class *WorkflowInstance* is handling one executed workflow created from a previously defined *TaskWorkflow*, which is referenced via the *workflow*-property. A *WorkflowInstance* is not only created from a *TaskWorkflow*, but also for each *SubProcesses* of the *TaskWorkflow* a new *WorkflowInstance* is created. *WorkflowInstances* instantiated from *SubProcesses* have to reference the *WorkflowInstance* of their *TaskWorkflow* as *parent* property. The *WorkflowInstance* of the *TaskWorkflow* does not have a parent. Each *WorkflowInstance* is additionally attributed with a *currentActivity* and *currentTransition* referencing the currently processing or waiting to be executed *ActivityInstance*, respectively the last *TransitionInstance* taken. Finally, each *WorkflowInstance* can have additional variables, which can be used by *ActivityInstance* for computations.

3.2.5.2 *ActivityInstance*

An *ActivityInstance* is an instantiated *Activity* consisting of *incomingTransitions* and *outgoingTransitions* as well as an *activityBehavior*, which implements the functionality of the referenced *Activity*. Additionally, each *ActivityInstance* contains *variables*, which can be used in the computations of the *Activity*.

3.2.5.3 *TransitionInstance*

The *TransitionInstance* represents the instantiated *SequenceFlow* between two *ActivityInstances*.

3.2.6 Resource Management

The *resource management* package (see Fig. 3.13) is responsible for handling both, human and non-human resources within iVolunteer. All

users subscribed to the marketplace component should be able to add their resources, they can bring in, especially for non-human resources like, benches or tables. Afterwards, tasks are able to claim the resources for the time-frame, they are executed.

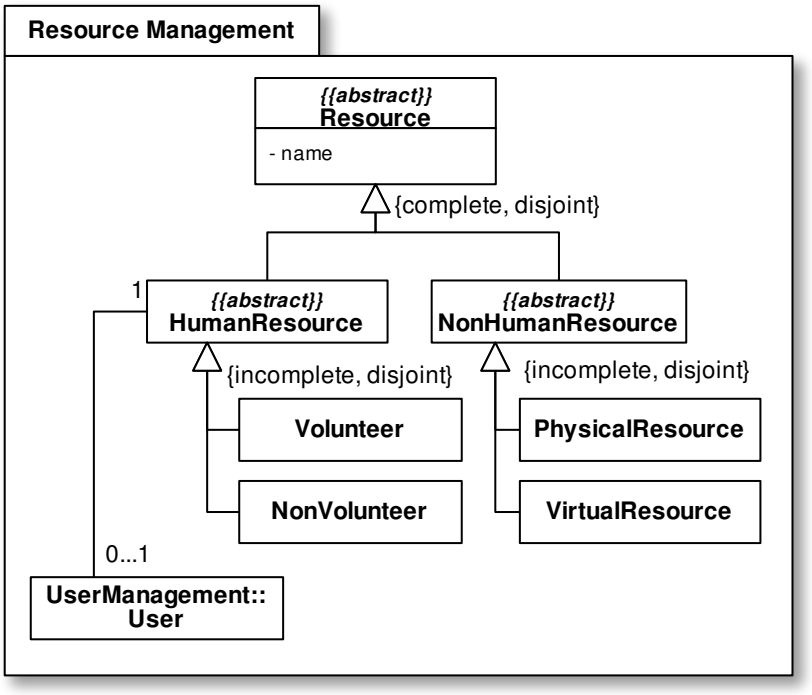


Figure 3.13: Resource Management Package

3.2.6.1 *Resource*

A *Resource* can be described as a source or supply from which a benefit is obtained [7]. *Resources* are generally divided into either *HumanResources* or *NonHumanResources*, which will be described in the following.

3.2.6.2 *HumanResource*

Within iVolunteer, *HumanResources* are further divided into *Volunteers* and *NonVolunteers*. While the former describes volunteers of a VIO, the latter describes paid employees of the VIO (e.g., help-seekers). Both, *Volunteers* and *NonVolunteers* can be linked to registered *User* of the marketplace. In order to also allow for the management of *Volunteers* and *NonVolunteers*, which are not yet registered to iVolunteer, a reference to the *User* is not necessary.

3.2.6.3 *NonHumanResource*

In contrast to *HumanResources*, *NonHumanResources* are not inseparable from human individuals. They can further be divided into either *PhysicalResources* and *VirtualResources*. While the former describes *NonHumanResources* that are tangible, the latter are intangible.

3.2.7 Recommendation Management

The *recommendation management* package (see Fig. 3.14) offers ways to recommend or suggest *Tasks* to volunteers. It differentiates between *Recommendations* and *Suggestions*, where the former is triggered by the iVolunteer system automatically, while the latter is triggered by another user (e.g., a volunteer suggesting a task to a friend). A further difference is, that in order to compute *Recommendations*, appropriate matching algorithms have to be applied to find fitting tasks for volunteers.

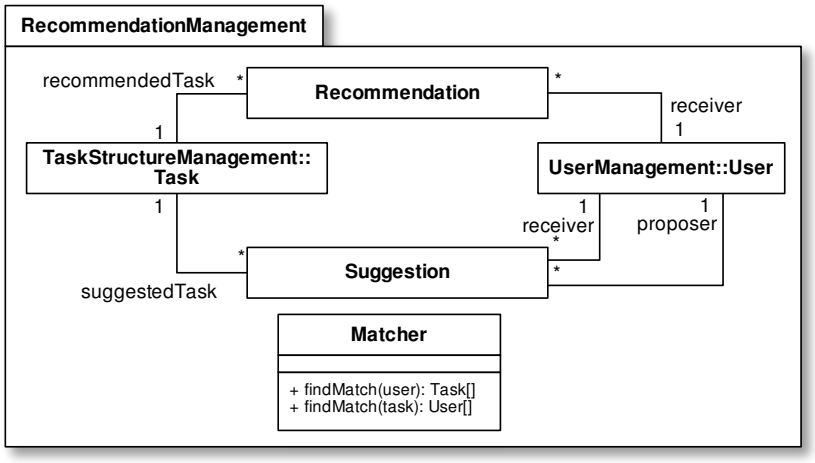


Figure 3.14: Recommendation Management Package

3.2.7.1 *Recommendation*

A *Recommendation* is a proposal by iVolunteer to a volunteer (i.e., the *receiver*), containing a *recommendedTask* that matches the published *Achievements* of a volunteer (i.e., *Achievements* in the *PublicVolunteerRepository*) as best as possible. In order to find such fitting tasks, the class *Matcher* is used (see Sect. 3.2.7.3).

3.2.7.2 *Suggestion*

A *Suggestion* is a proposal by another user (i.e., the *proposer*) to a volunteer (i.e., the *receiver*) containing a *suggestedTask*.

3.2.7.3 *Matcher*

The *Matcher* is the basis for finding appropriate *Tasks* for volunteers and vice versa. It, therefore, contains two methods - *findMatch(user)* for the former and *findMatch(task)* for the latter. In order to find fitting tasks for volunteers respectively fitting volunteers for tasks, the *Matcher* has to implement algorithms belonging to the domain of recommender systems. The most widely used recommendation techniques are content-based recommendation and collaborative filtering [25], which will be explained in the future work section of this thesis (see Sect. 6.2.3.1).

3.2.8 Achievement Management

The *achievement management* (see Fig. 3.15) package handles earned achievements for volunteers. The main class within the achievement management is *Achievement*, which is either an *AchievedCompetence* or a *CompletedTask*. Compared to the already described classes *Competence* and *Task*, which are primarily used to describe tasks and competences, the classes *AchievedCompetence* respectively a *CompletedTask* state that a volunteer earned a *Competence* or completed a *Task*. While currently only those two classes represent possible *Achievements*, it is considered to include further types of achievements like ranks, badges or awards in the future (see Sect. 6.2.1.1).
Achievements form the main element of the volunteer footprint and must therefore be synchronizable between the local repository of the volunteer and the marketplaces. Thus, as already stated in Sect. 3.2.1, the class *Achievement* is required to inherit from *HashableObject*, since *Achievements* are hashed and stored in the blockchain for verification purposes.
The achievement management further introduces the *PublicVolunteerProfile*, which contains all *Achievements*, that are published at the respective *Marketplace*. Later in the description of the local repository package (see Sect. 3.3.1), the *PrivateVolunteerProfile* is introduced, containing all *Achievements* that are synchronized to the decentralized storage of the volunteer, i.e., their local repository. A detailed overview of the interconnection between those two "volunteer profiles" will be discussed there.

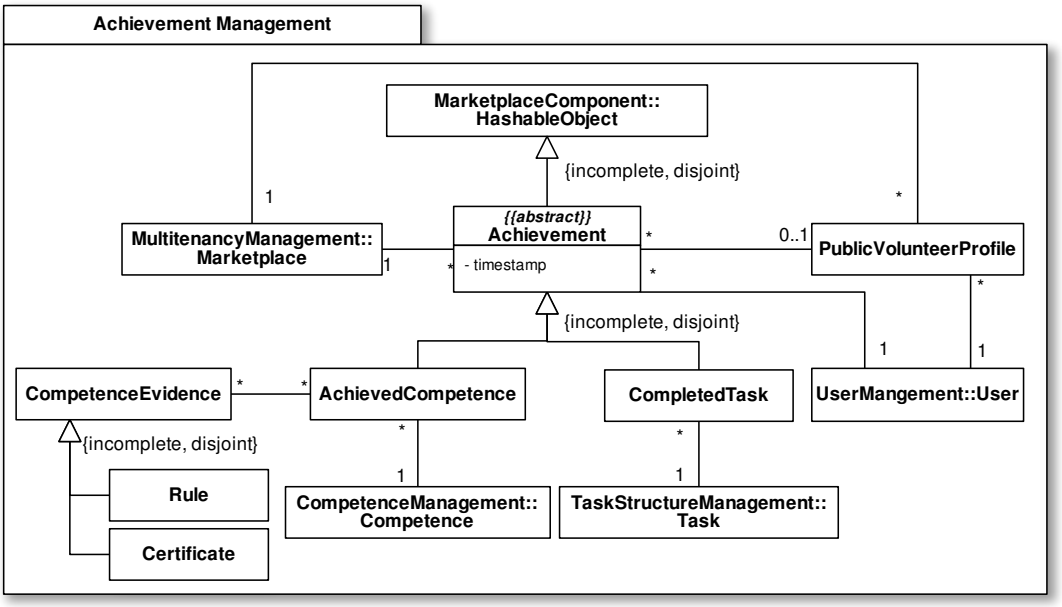


Figure 3.15: Achievement Management Package

3.2.8.1 *Achievement*

The abstract class *Achievement* inherits from *HashableObject* and is the superclass of all sorts of accomplishments a marketplace offers, like the acquirement of a competence for one volunteer (i.e., *AchievedCompetences*) or the completion of a task by a volunteer (i.e., *CompletedTasks*). Each *Achievement* references the *User*, obtaining the achievement, the *Marketplace* issuing the achievement and the time when the achievement was issued as *timestamp*. After an *Achievement* is issued (i.e., created because a volunteer earned a competence), it is initially neither part of the *PublicVolunteerProfile* nor of the *PrivateVolunteerProfile*. First, the volunteer has to synchronize the *Achievement* into their *PrivateVolunteerProfile*, afterwards, they can publish it and thus adding it to their *PublicVolunteerProfile*.

3.2.8.2 *CompletedTask*

The *CompletedTask* marks that a volunteer finished working on a certain *Task*. It, therefore, credits the volunteer for participating in a task. For each *Task* that is carried out by one or more volunteers, each of them receives a *CompletedTask* achievement when the task life-cycle is finished.

3.2.8.3 *AchievedCompetence*

Similar to the *CompletedTask*, representing, that a volunteer completed a task, an *AchievedCompetence* indicates that a volunteer acquires a competence.

3.2.8.4 *CompetenceEvidence*

A *CompetenceEvidence* is necessary in order to state how an *AchievedCompetence* was awarded and is therefore necessary for the traceability of the competences a volunteer earned. *CompetenceEvidence* is either a *Rule* that states how the competence was awarded or a *Certificate* proving that a volunteer acquired the competence.

3.2.8.5 *Rule*

A *Rule* gives information about how a volunteer earned a *Competence* and defines which preconditions must be fulfilled in order to receive which *Competence*. The definition of preconditions allows for several degrees of freedom and can involve rules regarding the completion of one or several specific tasks, the time a volunteer already contributed, a set of other competences a volunteer must already have, etc. Rules are the primary way for volunteers to acquire *Competences* within iVolunteer.

3.2.8.6 *Certificate*

A *Certificate* proves that a volunteer already acquired a *Competence*. *Certificates* are used in order to allow volunteers to bring in *Competences* they acquired outside of iVolunteer.

3.2.8.7 *PublicVolunteerProfile*

The *PublicVolunteerProfile* represents the set of *Achievements* a volunteer disclosed at a *Marketplace*. In contrast, the *PrivateVolunteerProfile* (see Sect. 3.3.1.1) consists of the set of *Achievements* a volunteer synchronized into the local repository. This differentiation is subtle but not less important because the sets of *Achievements* contained in the *PublicVolunteerProfile* and *PrivateVolunteerProfile* can be different. Furthermore, it is important to consider, that a marketplace can only use *Achievements* that are contained in the *PublicVolunteerProfile* because it does not have access to any Achievements of the *PrivateVolunteerProfile*. In Sect. 3.3.1.1, application scenarios of the interconnection between the *PublicVolunteerProfile* and *PrivateVolunteerProfile* will be discussed.

3.3 FOOTPRINT COMPONENT

As already stated in Sect. 2.4, the *footprint component* has two main responsibilities - (i) *footprint synchronization management* and (ii) *footprint verification management*. While the former is responsible for the synchronization of the volunteer footprint between the local repository and the marketplaces, the latter guarantees that the entries of the volunteer footprint can be verified in order to detect inconsistencies, e.g., when volunteers erroneously add competences to the local repository that they did not earn. From those two responsibilities, the packages *local repository* and *trust management* arose realizing them. The local repository represents the storage of the volunteer footprint, that is solely operated by the volunteer and therefore is the counterpart to the *PublicVolunteerProfile* discussed in Sect. 3.2.8. The trust management allows for the already addressed verification of the footprint by employing a blockchain.

3.3.1 Local Repository

The *local repository* package (see Fig. 3.16) contains the already mentioned *PrivateVolunteerProfile* comprising all *Achievements* a volunteer has synchronized. In contrast to the *PublicVolunteerProfile* which exists for each marketplace, the *PrivateVolunteerProfile* exists exactly once per volunteer and is stored decentralized in the local repository, accessible only by the respective volunteer. The local repository cannot be accessed by a marketplace or any other user without the approval of the respective volunteer, possessing it. Whenever volunteers wants to synchronize the volunteer footprint between the local repository and one marketplace, access to the local repository for the marketplace has to be granted.

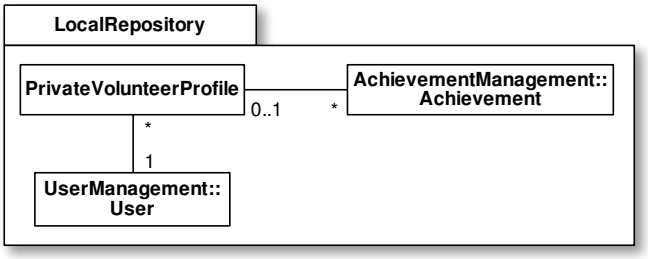


Figure 3.16: Local Repository Package

3.3.1.1 *PrivateVolunteerProfile*

The class *PrivateVolunteerProfile* contains a set of *Achievements*, that a certain volunteer already acquired (see Sect. 3.2.8). The primary usage of the *PrivateVolunteerProfile* is to allow for the synchronization of these *Achievements* between the local repository and the marketplaces' *PublicVolunteerProfile*.
In the following, scenarios regarding the interconnection between the *PublicVolunteerProfile* of a certain marketplace and the *PrivateVolunteerProfile* are discussed. Overall, three different cases wrt. the content of the *PublicVolunteerProfile* and the *PrivateVolunteerProfile* can be distinguished. The first case is where the *PrivateVolunteerProfile* and the *PublicVolunteerProfile* of one marketplace are completely distinct, i.e., they contain no common *Achievement*. In case two and three, the *PrivateVolunteerProfile* and the *PublicVolunteerProfile* overlap respectively are equal. In the subsequent Venn-Diagrams, the set A_{public} corresponds to the set of *Achievements* of the *PublicVolunteerProfile* while $A_{private}$ represents the set of *Achievements* of the *PrivateVolunteerProfile*.

DISTINCT VOLUNTEER PROFILES ($A_{private} \cap A_{public} = \emptyset$) In the first scenario, the set of *Achievements* of the *PublicVolunteerProfile* and the *PrivateVolunteerProfile* are distinct (see Fig. 3.17), i.e., they don't have any *Achievements* in common. This case is also prevailing if one volunteer profile does not contain any *Achievements*. In general, having distinct volunteer profiles is not desirable, neither for the marketplace nor for the volunteer, because volunteers cannot use the functionality of iVolunteer to its full extent. For example, it will not allow volunteers to reserve for certain tasks, which require some of those *Achievements* in order to be able to carry out these tasks.

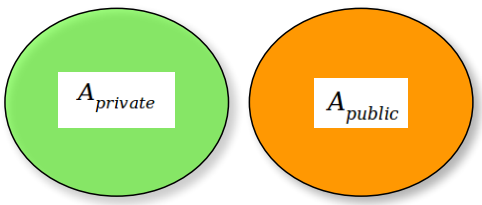


Figure 3.17: Distinct Volunteer Profiles

OVERLAPPING VOLUNTEER PROFILES ($A_{private} \cap A_{public} \neq \emptyset$) An overlapping *PublicVolunteerProfile* and *PrivateVolunteerProfile* (see Fig. 3.18) share at least one achievement. All *Achievements* of the *PublicVolunteerProfile* can therefore be used by the marketplace in order to suggest appropriate *Tasks* that are similar to previously accomplished *CompletedTasks* and *AchievedCompetences*. In order to increase the intersection between the *PublicVolunteerProfile* and the *PrivateVolunteerProfile*, volunteers have to publish *Achievements* (see Sect. 2.4.2) from their local repository (i.e., *PrivateVolunteerProfile*) to the *PublicVolunteerProfile* of the respective marketplace. If all *Achievements* from the *PrivateVolunteerProfile* are published, both volunteer profiles should contain exactly the same *Achievement* and are therefore equal (cf. below).

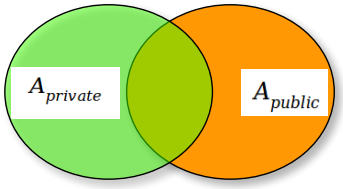


Figure 3.18: Overlapping Volunteer Profiles

EQUAL VOLUNTEER PROFILES ($A_{private} = A_{public}$) If both, the *PublicVolunteerProfile* and the *PrivateVolunteerProfile* contain the same set of *Achievements* (see Fig. 3.19), they are deemed equal. Having an equal set of *Achievements* across both, the *PublicVolunteerProfile* and the *PrivateVolunteerProfile* is ideal for the respective marketplace, because it can suggest more suitable *Tasks* to the volunteers. Nevertheless, a volunteer may strive not to have equal volunteer profiles, because of its possible repercussions if one marketplace and therefore its VIO knows all achievements of a volunteer from another marketplace (i.e., another VIO).

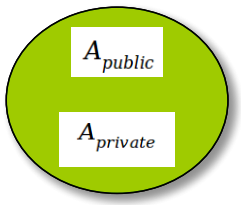


Figure 3.19: Equal Volunteer Profiles

3.3.2 Trust Management

The *trust management* package (see Fig. 3.20) realizes the footprint verification management (see Sect. 2.4) and lays the foundation for the *immutability*, *trustability*, *verifiability* and *information obfuscation* characteristics of the volunteer footprint entries. The immutability characteristic is achieved by incorporating a blockchain (see Sect. 4.1.4), storing *HashableObjects* and thereby establishing trust between the volunteers and

the marketplaces and allowing for their verifiability. Last, the *HashableObjects* in the blockchain are stored as hashes thereby achieving information obfuscation. More precisely, *Contracts* are created from the *HashableObjects*, thereby obfuscating the information of the *HashableObjects*, while still guaranteeing that the original *HashableObjects* remain verifiable by hashing them and comparing the hash with the hash of the *Contracts*. Therefore, the trust management is primarily responsible for (i) creating, (ii) inserting and (iii) verifying blockchain entries. These three functionalities are implemented by the classes *Hasher*, *Contractor* and *Verifier* respectively.

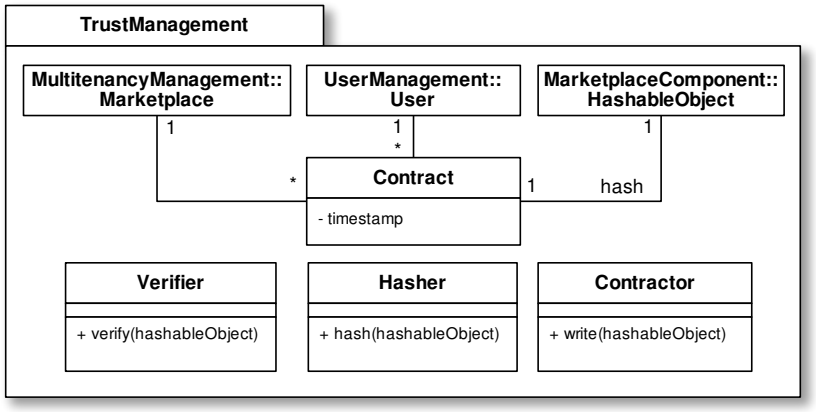


Figure 3.20: Trust Management Package

3.3.2.1 *Contract*

A *Contract* is an obfuscated (i.e., hashed) version of a *HashableObject*, which is represented as its *hash*. In addition, *Contract* comprises the *user*, it belongs to, the respective *marketplace* as well as a *timestamp* of its creation time. The mentioned obfuscation is required because the *Contracts*, which are stored in the blockchain, are accessible to other users of iVolunteer.

3.3.2.2 *Hasher*

The *Hasher* is responsible for creating one-way hashes from *HashableObjects* (see Sect. 3.2.1). The *hashes* are created by concatenating all properties of the *HashableObject* and applying the SHA-256 hashing algorithm.

3.3.2.3 *Contractor*

The *Contractor* provides means to insert *HashableObjects* into the blockchain by first hashing them and creating a *Contract* from the hash. The created *Contract* is then inserted into the blockchain.

3.3.2.4 *Verifier*

The *Verifier* is used to verify *HashableObjects* (see Sect. 3.2.1) (e.g., *Achievements* of a volunteer's local repository). The verification process involves the hashing of the *HashableObject* and looking up the resulting hash

in the blockchain by comparing whether an existing *Contract* in the blockchain has the same *hash*. If the blockchain does contain such a *Contract*, the *Verifier* returns true, otherwise false.

4 SYSTEM ARCHITECTURE

Contents

In this chapter, the system architecture of the developed proof-of-concept prototype - based on the conceptual approach - is discussed. First, each architecture component will be described, followed by a depiction of the concrete deployment of each component in a distributed way, showing the applicability of the distributed architecture for the proof-of-concept prototype.

4.1 ARCHITECTURE COMPONENTS

The architecture components comprise the (i) *cross-marketplace component* and the (ii) *marketplace component*, originating from the identically named packages of the conceptual approach, the (iii) *trustifier*, (iv) *blockchain* and (v) *local repository* derived from the footprint component and the (vi) *client* enabling different user roles (see Sect. 2.1) to use the functionality as stated in Chapter 2. Fig. 4.1 shows the architecture components, including the components provided and required interfaces, illustrating the data flow between the architecture components.

4.1.1 Cross-Marketplace Component

As described in Sect. 3.1, the *cross-marketplace component* is responsible for (i) *user management*, (ii) *multitenancy management*, (iii) *social management* and (iv) *GUI management*. The depicted cross-marketplace architecture component of the proof-of-concept prototype focuses only on the former two responsibilities and disregards the latter two because it would go beyond the scope of this thesis. While the social management will be discussed in the future work chapter (see Sect. 6.2.2.1) of this thesis, the *GUI management* will be discussed in-depth in the thesis

of my colleague Berthold Roiser.

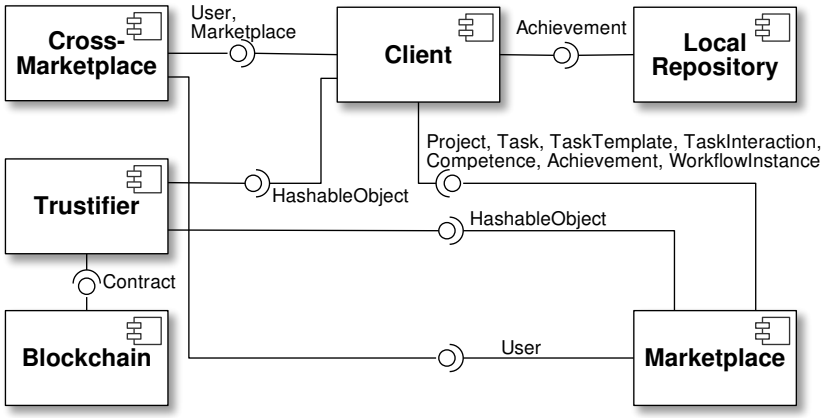


Figure 4.1: System Architecture Components

The cross-marketplace component is a stand-alone server application running and employing an own cross-marketplace database, storing both, *Marketplace* and *User* entities, as well as relationships between them, i.e., which *Users* are subscribed to which *Marketplaces*, carrying out a certain *UserRole* (see Sect. 3.1.1 resp. 3.1.2). Thus, the cross-marketplace component globally manages all users across all marketplaces and therefore is able to provide a single sign-on authentication for all marketplaces as stated in Sect. 2.3.1. All further user-specific information (e.g. achievements of volunteers, etc.) and interactions with the marketplaces are stored decentralized solely at the databases of the marketplaces themselves (see Sect. 4.1.2). As described in Sect. 2.3, the cross-marketplace component allows for users and marketplaces to initially perform registration at iVolunteer as well as manage subscriptions of users to marketplaces. Furthermore, it provides information about *Users* and *Marketplaces* via interfaces to the marketplace components and the clients as can be seen in Fig. 4.1. The cross-marketplace component is managed by the platform administrator (see Sect. 2.1.4), which is responsible for monitoring and troubleshooting issues regarding the cross-marketplace component.

4.1.2 Marketplace Component

Similar to the cross-marketplace component, the *marketplace component* is also a stand-alone web server application, which can run on a different server than the cross-marketplace component. In contrast to the cross-marketplace component, which runs just once, each VIO can operate multiple marketplace components distributed across multiple servers. Each marketplace component employs two databases - the *workflow database* and the *marketplace database*. While the workflow database stores all task life-cycle definitions and instances, the marketplace database stores all other objects as described in Sect. 3.1.2.1. This distinction is

necessary because a dedicated *workflow engine* (i.e., Activiti[1]) is utilized in order to handle the task life-cycles, relying on an own database, i.e., the *workflow database*. Similar to the cross-marketplace component which is managed by the platform administrator, each marketplace component is operated by its respective marketplace administrator (see Sect. 2.1.3).
The marketplace component provides marketplace functionalities for VIOs as described in Sect. 3.1.2.1 - (i) *task structure management*, (ii) *task behavior management*, (iii) *resource management*, (iv) *competence management*, (v) *achievement management* and (vi) *recommendation management*. Since a full implementation of the marketplace component would go beyond the scope of this thesis, the focus laid on a comprehensive implementation of the tasks structure and task behavior management, as well as a shallow implementation of the *achievement management* and *competence management*. Both, the resource management and the recommendation management will not be discussed in this chapter, yet especially the recommendation management will be discussed separately in the future work chapter (see Sect. 6.2.3.1). A special focus was laid on the *task behavior management* by realizing a configurable task life-cycle through the utilization of Activiti as workflow engine, allowing to define and execute workflows (see Sect. 3.2.4 and 3.2.5). Generally speaking, the task workflow prescribes the steps a task has to go through from creation to completion and therefore makes the task execution configurable. This is especially necessary because organizations typically have different requirements and rule-sets when it comes to their processes and executions of their tasks (see Sect. 2.2.2).
In order to allow the marketplace component access to the users of iVolunteer, respectively access to its own subscribed users it consumes the cross-marketplace component's *User* interface as shown in Fig. 4.1. Furthermore, the marketplace component provides interfaces for all the objects stored in both databases, i.e., *Project*, *Task*, *TaskTemplate*, *TaskInteraction*, *Competence*, *Achievement* and *WorkflowInstance* to the *clients*, thus allowing for them to be displayed to the users and enabling interactions with them (e.g., creating a new task). In order to store respectively verify these objects, the marketplace component consumes the *Trustifier's* interface for *HashableObjects*.

4.1.3 Trustifier

The *trustifier* implements the *trust management* as introduced in Sect. 3.3.2 and is responsible for managing access for clients and marketplace components to *Contracts* (i.e., obfuscated *HashableObjects*; see Sect. 3.3.2.1) stored in the blockchain. As shown in Fig. 4.1 the trustifier provides interfaces of *Contracts* to both, the clients (especially the volunteer clients) and the marketplaces by consuming the interface of the blockchain. Thus, the trustifier has to be able to (i) generate and write *Contracts* as well as to (ii) verify existing *HashableObjects* by generating a hash and comparing it with the hash of *Contracts* stored in the blockchain. While the proof-of-concept prototype allows all actors to read from and write to the blockchain, only the trustifier is able to write

1 https://www.activiti.org/

verifiable *Contracts* because only the trustifier knows the production rules of *Contracts*, i.e., how *Contracts* are created from *HashableObjects*. This behavior leads to two major drawbacks - (i) participants have to trust the trustifier component in order to generate *Contracts*, that both, the volunteer and the marketplace agree upon, thus diminishing the trust that is established by the blockchain and (ii) that the production rules have to be kept secretly, otherwise every participant of the blockchain could create valid blockchain entries, leading to the fraudulent behavior, where volunteers would be able to add verifiable competences to their footprint, they did not acquire beforehand. Both of these shortcomings, will be discussed in the Master's thesis of my colleague Philipp Starzer, where a solution for them will be presented, revolving around the realization of the trustifier functionality as smart contracts within the blockchain. However, in order to still balance trust between volunteers and the marketplaces as best as possible with the existing architecture, the trustifier is operated by neither of them, but instead by the platform administrator (see Sect. 2.1.4), which is also responsible for managing the cross-marketplace component. Therefore, both, the volunteers and the marketplaces should be restricted with their power over the other participants, when it comes to the generation and verification of *Contracts*.

4.1.4 Blockchain

A blockchain represents a distributed ledger, where most or all of the participants - which may be unknown - agree on the current state and all states beforehand of each blockchain asset (i.e. the data stored in the blockchain). The reason, why this is such a taxing task is because, due to the sheer unlimited number of potential participants and their possibly destructive intentions, blockchains still guarantee that a consensus between all or most of the participants is reached and can be trusted afterwards even though the participants do not have to trust each other, leading to the establishment of trust in a trust-less environment. Often times the terms blockchain and crypto-currencies are used synonymously, but to characterize a blockchain in general as a currency platform (e.g. Bitcoin [26]) does not nearly do the technology enough justice because nowadays blockchains are more sophisticated and are applicable not only in finance but rather in general-purpose applications [27]. Therefore, the main characteristics of blockchains stray further than the establishment of trust, but rather covers the (i) *immutability* of the blockchain assets, the (ii) incorporation of *smart contracts* and user-defined entities additionally or instead of coins as well as the (iii) usage of *interchangeable consensus mechanisms* for blocks and transactions. Especially the first characteristic - immutability - is needed for the footprint management because it guarantees that the footprint cannot be maliciously altered, e.g., volunteers appending competences that they do not have. There already exists a plethora of publications discussing the applicability of blockchain-driven verification systems in various different domains like in industry [28], to support supply chain management processes [29], [30], for educational purposes [31], [32] as well as generally for the verification of documents [33].

4.1.4.1 *Design Decision for a Blockchain*

As stated in the requirements for the *footprint verification management* (see Sect. 2.4.1), the following non-functional requirements are essential in order to guarantee that the verification of the footprint entries can be successfully performed, while still ensuring data privacy. These four requirements cover (i) the *trustability*, that verified footprint entries can be trusted by all participants, the (ii) *immutability* of the entries in the verification storage, guaranteeing that they are not changed afterwards, thus not allowing participants (e.g., volunteers and VIOs) to change stored entries after they were issued, thereby not permitting to either improve or worsen it illegitimately. Furthermore, (iii) *information obfuscation* is necessary in order to protect private data and last (iv) *verifiability* has to be ensured even if the information is obfuscated.
Since one of the main features of blockchains is to ensure trust in a trustless environment it completely fulfills the first requirement better than any substitute because all other possible solutions for a verification storage (e.g. centralized database operated by the government) would at least require some sort of trust in the organization managing the verification storage. As for a blockchain-based solution, the trust is established by design through distributing the blockchain to several or a nearly unlimited amount of participants, while still ensuring that a consensus between the participants can be found. While in theory, blockchains are not completely foolproof, when it comes to trusting them (e.g. through majority attack), the threshold to deliberately change the assets with malicious intent is practically not possible to reach as long as there are enough participants holding replications of the blockchain. The second requirement is met because all transactions of a blockchain are stored in blocks which are cryptographically linked together, thereby enforcing immutability of the transactions. Since the transactions are immutable, it can be ensured, that changes to the blockchain assets, which can only be performed through these transactions, can be detected. Thus, immutability of the blockchain assets is realized twofold - (i) by being able to detect all changes and (ii) by being able to only offer transactions that disallow the change of existing blockchain assets. While the second way to ensure immutability can be also be met with solutions other than a blockchain used as verification storage, the detection of changed transactions represents a major reason to prove the immutability of blockchain assets through the immutability of the transactions. The third and fourth requirement - *information obfuscation* respectively *verifiability* - are realized by storing the footprint entries as *Contracts*, veiling the contained personal information, but still allow for verification because the obfuscation was fulfilled by generating a one-way hash, which allows for verification, if the original footprint entry is hashed as well and compared with the hashes of the blockchain assets.

4.1.4.2 *Design Decision for utilizing Hyperledger Fabric*

Hyperledger Fabric[2] (HLF) is a modular and extensible open-source blockchain platform for deploying and operating a permissioned blockchain

2 www.hyperledger.org/projects/fabric

[34] and is designed to allow for pluggable components (e.g. different consensus mechanisms). The central component of HLF is the distributed ledger, consisting of two different databases - the *world state database* and the *transaction log*. While the world state database contains the current state of all *assets*, the participants of the blockchain network agree on, the transaction log comprises all *transactions* that were executed in order to create and change these assets. Transactions comprise of *chaincode* (i.e., smart contracts), allowing for the utilization of user-defined transactions which can be invoked by participants of the blockchain network. *Transactions* are ordered by the *ordering service*, responsible for enforcing consensus, thereby generating blocks that are persisted on the *peers*, which hold decentralized replication of the distributed ledger. In order to simplify and accelerate application development, a development toolset and framework called *Hyperledger Composer* is provided, serving as an easy-to-use interface between applications and Hyperledger Fabric. Thus, *Hyperledger Composer* functions as a high-level business abstraction layer to the Fabric network. Hyperledger Composer allows to easily model a blockchain network, its participants, the assets that should be stored within the blockchain and the definition for the transactions, enabling participants to instantiate respectively change existing assets. This paragraph should only give a very brief overview of both, Hyperledger Fabric and Hyperledger Composer and their major components because describing their full extent would go far beyond the scope of this thesis and will be covered more in-depth by the Master's thesis of my colleague Philipp Starzer.
Following this short introduction to HLF are the reasons why it is used as blockchain platform for the proof-of-concept prototype. First of all, HLF is not only developed by a major company (IBM) but also already used by big players, such as American Express, SAP, Intel and many more, leading to frequent releases of new features. Second, HLF does fulfill all minimum requirements, such as definable assets, as well as the support for smart contracts, which is necessary to create contracts and verify them. Third, since iVolunteer does not need to support cryptocurrency in any way which is used in public blockchains in order to regulate the number of transactions processed from participants (e.g. gas price for smart contract invocation within Ethereum [35]), the decision to use a private blockchain over a public one was made.

4.1.5 Local Repository

The *local repository* represents the volunteer's *private volunteer profile*, storing their obtained achievements, i.e., finished tasks and achieved competences (see Sect. 3.3.1). The private volunteer profile is manifested as JSON-file (see Fig. 4.1), allowing for extensibility in the future, when further types of footprint entries are realized. By storing the footprint entries within the private volunteer profile in the same format as the public volunteer profiles, both, footprint entries from the marketplace and the local repository can use the same hashing function of the trustifier, without having to apply data transformation beforehand. A simple existing open source JSON-server implementation[3] is employed,

3 https://www.npmjs.com/package/json-server

allowing volunteers to give marketplaces access to their private volunteer profile by providing REST-endpoints for the whole JSON-file respectively for single footprint entries. Since one of the requirements of iVolunteer is to give volunteers sovereignty over their data, they can choose at any time whether they want to publish their local repository either by starting or stopping the application. However, in order to carry out synchronization between the marketplaces and the local repository, it is a necessity for the respective volunteer to publish the private volunteer profile by starting the application. Without starting the application, thus explicitly agreeing with the synchronization, marketplaces cannot directly change the content of the local repository.

Listing 4.1: Example of a Local Repository

```
{
  "repository": [
    {
      "id": "1",
      "volunteer": {
        "username": "pstarzer"
      },
      "finishedTasks": [
        {
          "id": "5af7e2ee379bb3593ceb041b",
          "taskId": "5af7e28a379bb3593ceb0415",
          "taskName": "Task 1",
          "taskDescription": "Task 1 Description",
          "marketplaceId": "Marketplace1",
          "timestamp": 1526194926350
        }
      ],
      "achievedCompetences": [
        {
          "id": "939c6ca3-1375-424e-816a-369e04465384",
          "competenceId": "5af7e227379bb3593ceb0412",
          "competenceName": "Flexibility",
          "marketplaceId": "Marketplace1",
          "timestamp": 1526194926350
        }
      ],
    }
  ]
}
```

4.1.6 Client

The frontend of iVolunteer is designed as an Angular[4] 6 web application, providing different GUI views for users and their respective roles on the marketplaces - volunteer, help-seeker, marketplace administrator and platform administrator. Each view offers different interaction possibilities with the cross-marketplace and the marketplaces, e.g., managing tasks for help-seekers or viewing achievements for volunteers. In order

4 https://angular.io/

for users to interact not only with one marketplace but with several, the client provides an aggregated view, allowing users to view all tasks, achievements, etc. of all subscribed marketplace within the same view, thus heightening user experience through allowing users to interact with all marketplace within the same view. Additionally, filtering options are provided, allowing users to select a subset of their subscribed marketplaces, thereby filtering the aggregated view accordingly.

4.2 ARCHITECTURE DEPLOYMENT

After explaining each individual component in the previous section, this section revolves around the concrete deployment of these components as can be observed in Fig. 4.2. The deployment diagram shows the structure of the prototype's run-time system, including the distributed servers, executing the components and the communication paths between them. These interactions between the components take place via XMLHttpRequests (XHR) utilizing the published REST endpoints for the entities displayed in the component diagram (see Fig. 4.1).

4.2.1 Cross-Marketplace Server

The *Cross-Marketplace Server* hosts the Angular web application, as well as the *Trustifier* and the *Cross-Marketplace* components. Typical for an Angular web application, the respective files (JavaScript, HTML, CSS, etc.) files are bundled together using the webpack[5] module bundler, thus compressing the required files and decreasing loading times for the web application because less data has to be transferred from the cross-marketplace server to the clients. Both, the *Trustifier* and the *Cross-Marketplace* component are realized as Java Spring Boot[6] 1.5 server application, providing REST (Representational State Transfer) endpoints [36], which are accessed by the other components through XMLHttpRequests (XHR). To provide full CRUD (create, read, update, delete) functionalities for objects (e.g., User, Marketplace, HashableObject) that can be exchanged via the interfaces as displayed in Fig. 4.1, for each of the HTTP-methods - GET, PUT, POST and DELETE - a REST-endpoint is published.
Additionally, the cross-marketplace component manifests its objects into the *Cross-Marketplace Database*, mainly including registered Users and Marketplaces, as well as subscription of users to marketplaces and their respective roles they carry out. The cross-marketplace database is realized, employing the the NoSQL database - MongoDB[7], manifesting the objects in JSON format. Using a NoSQL database allows for simple extensibility of the stored objects because no common, explicit schema is enforced.

5 https://webpack.js.org/
6 https://spring.io/projects/spring-boot
7 https://www.mongodb.com

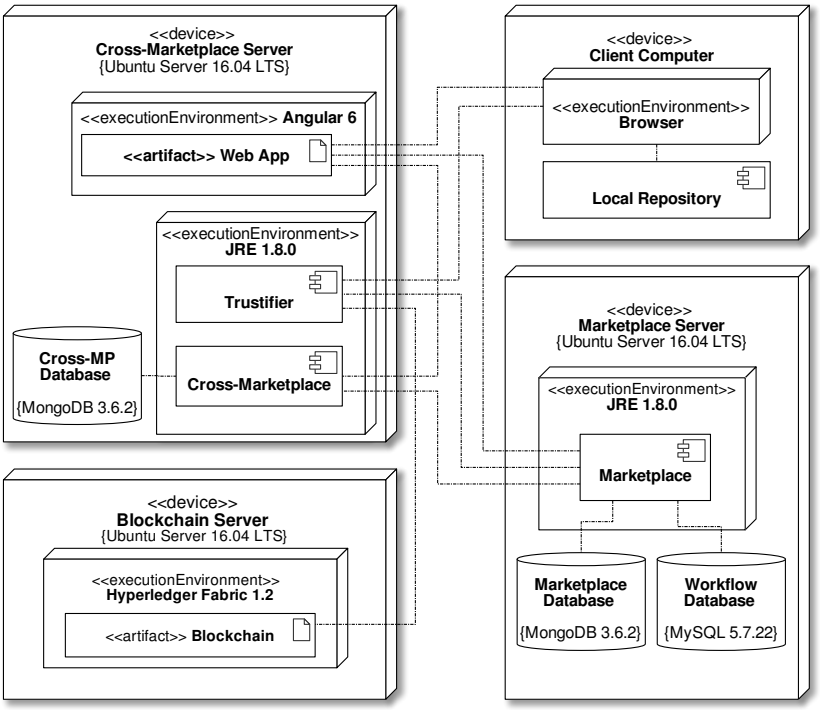


Figure 4.2: Deployment diagram of the iVolunteer application

4.2.2 Blockchain Server

The *Blockchain Server* hosts a minimal deployment of Hyperledger Fabric including one peer storing the distributed ledger and one ordering node, responsible for consensus generation. While for later production usages, a more sophisticated Fabric network would be necessary, including several peers and ordering nodes distributed and managed by different organizations, the focus for the proof-of-concept prototype laid on showing the applicability of the whole architecture and the interactions between the components, for which a sophisticated Fabric network is not needed. An in-depth discussion about blockchains, HLF and the integration into iVolunteer will be contained in the Master's thesis of my colleague Philipp Starzer.

4.2.3 Marketplace Server

The *Marketplace Server* publishes the marketplace component and its respective databases - the marketplace db and the workflow db. While the cross-marketplace component is designed as central communication broker, thus only have to run once (disregarding potential load balancing and fail-safe challenges), one marketplace component should exist for each VIO. Therefore, not only one marketplace server is considered as part of the system architecture, but numerous. The marketplace server is designed to run distributed, allowing VIOs to run marketplaces within their very own server infrastructure and allowing them to store their data therein, mitigating data centralization and the emer-

gence of a data silo. The usage of two databases per marketplace is owed to the workflow management framework Activiti[8], which depends on a MySQL[9] database in order to store workflow definitions and their respective instantiations. All other objects are stored in the marketplace database, including *Tasks*, Projects, *Achievements*, etc.

4.2.4 Client Computer

The client computer represents the user's device used to interact with the iVolunteer platform. The web browser running on the client computer downloads the Angular web application and allows for authentication and utilization of the functionalities described in Chapter 2. The client computer further hosts the local repository application, allowing to publish the private volunteer profile and grant the client access to footprint entries thereof.

8 https://www.activiti.org/
9 https://www.mysql.com/

5 RELATED WORK

Contents

After having described requirements, conceptual approach and architecture of iVolunteer, this thesis is rounded up in the following by providing a brief comparison of iVolunteer to a few selected existing VMS. A plethora of volunteer management systems already exists, mostly focusing on satisfying the needs of VIOs and supporting them with the management of volunteers and tasks, lacking means for volunteers to privately digitize and exploit their achievements, e.g., task accomplishments or earned competences. This may decrease engagement, since appreciation of volunteer work is the only reward available, and hinders the exploitation of engagement assets between NPOs and beyond. iVolunteer however, puts volunteers in the middle of concern by giving them sovereignty over their data and allowing for their usage throughout multiple VIOs.
In this chapter, a review and comparison of a few volunteer management systems with iVolunteer is conducted. First, the feature categories and their respective features are discussed, followed by a short introduction to a small range of carefully selected VMS - Freiwillig[1], Samaritan[2] and Volunteering Matters[3]. Afterwards, they will be compared on the basis of the presented features.

5.1 VMS FEATURE CATEGORIES

In order to compare iVolunteer with existing volunteer management systems, four feature categories are proposed, which are based on the requirements in Chapter 2, a previously conducted survey [7] as well as our own experience while developing iVolunteer. While the referenced survey compares the VMSs on a much more detailed level, the

1 http://www.freiwilligenweb.at
2 http://www.samaritan.com
3 https://volunteeringmatters.org.uk/

following features categories are primarily selected to showcase issues and drawbacks of currently available VMS in areas, where iVolunteer shines - (i) organization management, (ii) task management, (iii) footprint management and (iv) social aspects, all of which are going to be explained in the next sections.

5.1.1 Organization Management

The *organization management* is related to how well a VMS can depict the structure of VIOs. Naturally the organization management of formal and informal VIOs are different, not least because formal VIOs (e.g. Red Cross) are in general more strictly regulated wrt. their organizational structure than informal VIOs. The only feature of the organization management is the support of *multitenancy*.

5.1.1.1 *Multitenancy*

Whether a VMS supports multiple tenants or not is crucial, when it comes to allowing VIOs to setup and configure their projects and tasks. Not only whether multitenancy is integrated, but also in which ways VIOs can manage and configure certain entities and processes within the VMS is evaluated. Depending on the VMS, multitenancy can vary from entering basic information about a VIO to completely enabling a VIO to customize their organization, tasks or projects not only structurally, but also on a functional level.

5.1.2 Task Management

Task management comprises task-specific features including the configuration possibilities of tasks, task matching, as well as whether tasks can involve the utilization of non-human resources additionally to volunteers.

5.1.2.1 *Task Configuration*

Based on the task configuration requirement (see Sect. 2.2.2.1), the four task configuration possibilities are considered - (i) *Task Properties*, (ii) *Task Life-Cycle Configuration*, (iii) *Inter-Task Structural Relationships* and (iv) *Inter-Task Behavioral Relationships*.

5.1.2.2 *Task Matching*

Task matching refers to the process of finding adequate volunteers for a task respectively appropriate tasks for a volunteer. The problem of matching tasks and volunteers belongs to the domain of recommender systems and there exists numerous approaches to how the matching can be performed [37] (e.g., collaborative filtering, content-based or ontology-based), this feature will only consider, whether matching is integrated or not.

5.1.2.3 *Non-human resources*

The feature *non-human resource* relates to whether a VMS allows to integrate non-human resources and how flexible they can be managed and used. Since non-human resources often times are critical for the successful execution of tasks, their management and handling make up an important aspect of the task management. Examples for non-human resources are chairs or benches, which can be provided for a certain task.

5.1.3 Footprint Management

Footprint management is concerned with how volunteers are able to manage achievements, they earned during their voluntary activities and use them for other VIOs or elsewhere, e.g., for job applications. Footprint management is split into the following four features, essential to organizing a volunteer's footprint: (i) data sovereignty, (ii) immutability, (iii) synchronization and (iv) verification.

5.1.3.1 *Data Sovereignty*

Data sovereignty is related to whether the volunteers are solely storing their volunteer footprint or if it is stored centrally at the respective VIOs or the government.

5.1.3.2 *Immutability*

Important for trust into the volunteer footprint is, whether volunteers or VIOs can unjustifiably alter existing achievements or append new ones. Immutability is important, whether the verification of the footprint can be trusted.

5.1.3.3 *Synchronization*

Synchronization revolves around the fact, whether the footprint of each volunteer can be updated and synchronized. This feature does not only cover the insertion of new achievements, but also covers the update of already existing achievements when they are changed, e.g., when a competence expires after a certain time frame or when it is refreshed.

5.1.3.4 *Verifiability*

Last, it is important to be able to *verify* entries of the footprint in order to state, whether they are valid or not. Verification typically involves checking, whether the respective footprint entry is valid and was indeed issued by the specified VIO. While an offline check by calling the respective VIO would be always possible, this feature only considers verification through the volunteer management system.

5.1.4 Social Aspect

According to a study of Statistics Canada[4], almost half of all asked volunteers stated, that a main reason why they volunteer is, that their friends are engaged in the same organization as well [14]. Due to that, VMSs have to not only provide functionalities revolving around the task management, but also support the social aspects in various forms. Similar to the requirements for the social management (see Sect. 2.3.3) the following three features are distinguished: (i) *social relationship management*, (ii) *social awareness management*, (iii) *social communication management*.

5.2 VOLUNTEER MANAGEMENT SYSTEMS

For the sake of comparability, the presented volunteer management systems will be differentiated into VMS-portals and VMS-platforms. While VMS-portals are lightweight VMS with the main focus on mediating between VIOs and volunteers, VMS-platforms are fully-fledged stand-alone applications providing full support for volunteer and organizational management. iVolunteer falls into the category of VMS-platforms.
The following three systems were chosen, because all of them are web-based VMS-portals or VMS-platforms with main characteristics that fit well with the feature categories that were established beforehand. "*Freiwillig*" is a VMS-portal, that incorporates a very similar idea of a volunteer footprint primarily focuses thereon. *Samaritan* is a customizable VMS-platform with focus on volunteer recruitment ('eRecruiter'), organization management and task management ('eCoordinator'). Last, *Volunteering Matters* is another VMS-portal, focusing primarily on finding appropriate volunteers for voluntary activities.

5.2.1 "Freiwillig"

"Freiwillig[5]" is a VMS-platform operated by the Austrian Federal Ministry of Labour, Social Affairs, Health and Consumer Protection[6]. The main idea behind this platform is to incentivize cooperation of voluntary organizations and to inform about voluntary work or events not only in Austria, but around the globe as can be seen in Fig. 5.1.

4 https://www.statcan.gc.ca/
5 http://www.freiwilligenweb.at/
6 https://www.sozialministerium.at

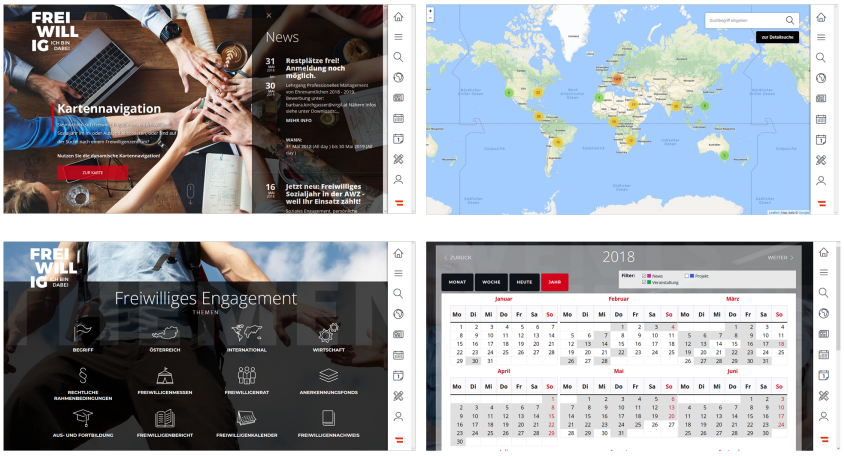


Figure 5.1: Impressions of "Freiwillig"

The main unique selling proposition of "Freiwillig" is that it enables volunteers to receive their volunteer footprint in the form of a volunteer's pass, which is in its essence a document listing all organizations a volunteer participates and a set of skills and competencies they obtained during their voluntary activities. The volunteer's pass is a hand-written document belonging to the volunteers themselves and is handed out by the Austrian Federal Ministry of Labour, Social Affairs, Health and Consumer Protection as well. This represents a major drawback compared to iVolunteer, which digitizes the voluntary engagement and allows for straightforward synchronization of the achievements into the local repository. By only allowing a hand-written document serving as the volunteer footprint, the overhead for managing the volunteer's pass may exceed the workforce of the VIO, thus will not be managed at all.

5.2.1.1 *Footprint Management*

For "Freiwillig", the volunteer's pass functions as volunteer footprint, but due to the fact that is is a hand-written documents both features *immutability* and *synchronization* are not supported.
Furthermore, the verification of the footprint turns out to be difficult, because the organization would have to verify each footprint entry by cross-checking it with their internal system, which is probably not automated nor generalized across multiple VIOs. *Data sovereignty* is provided, because the volunteers themselves own the volunteer footprint.

5.2.1.2 *Organization Management*

"Freiwillig" does support the functionality of *multitenancy* through allowing VIOs to register and manage their activities and events. However, since each VIO has to be registered with an extract of either their company registration or their association registration, informal organizations, like neighbourhoods or unregistered organizations are prohibited from using the platform.

5.2.1.3 *Social Integration*

"Freiwillig" does not support any social aspects, except a global timeline allowing VIOs to provide important news about activities and events.

5.2.1.4 *Task Management*

While "Freiwillig" does allow VIOs to manage events and projects, a dedicated task management is not integrated.

5.2.2 Samaritan

Samaritan[7] (see Fig. 5.2) is a VMS-platform, with the main focus on managing volunteer processes and lays a focal point especially on customization thereof. Thus, Samaritan targets especially large VIOs with their very own voluntary processes, which can be realized therein. Samaritan offers three central features: (i) volunteer recruitment support, (ii) volunteer management and (iii) volunteer tracking.

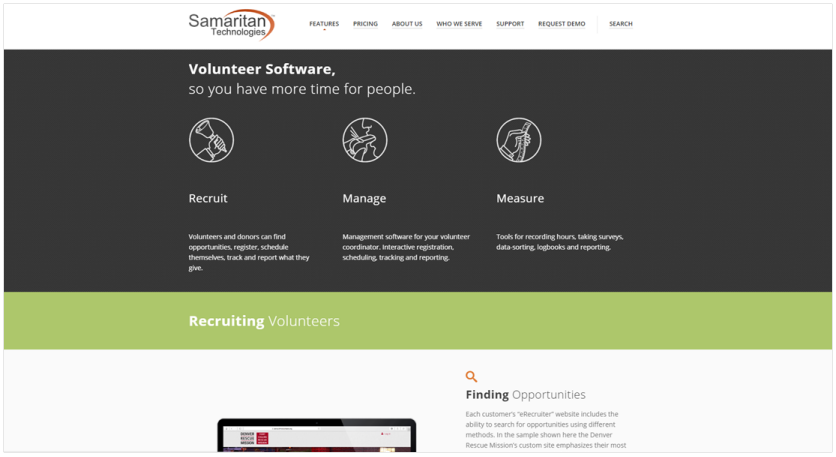


Figure 5.2: Impressions of Samaritan

VOLUNTEER RECRUITMENT SUPPORT Samaritan helps with the recruitment of new volunteers by providing a wide range of internet recruitment support (i.e. Facebook Integration, pre-built opportunities and registrations forms, calendar and map integration, etc.) as well as performing criminal background checks.

VOLUNTEER MANAGEMENT Volunteer management includes the opportunity to hierarchically structure a VIO's organization, scheduling volunteers with an calendar, automatic and manual e-mail notifications and a reporting system for voluntary activities.

VOLUNTEER TRACKING Samaritan supports an integrated data management, which tracks volunteers throughout their work by storing their activities as well as their time spent on them.

7 https://samaritan.com/

5.2.2.1 *Footprint Management*

Samaritan doesn't support footprint management at all, thus all information of the volunteer's work is stored centrally at the VIO and cannot be automatically used by the volunteers for other organizations.

5.2.2.2 *Organization Management*

Samaritan does support multiple tenants. Each tenant, i.e., a VIO is registered at the Samaritan web application.

5.2.2.3 *Social Integration*

Samaritan itself doesn't support any social interaction like chat, groups etc., however, volunteers can post activities on social media platforms like facebook or twitter.

5.2.2.4 *Task Management*

Within Samaritan, most of the features of a task management are provided, including the support for *structural task relationships*, *task configuration*, *workflow execution*, *task assignment* and *task matching*. As far as could be evaluated, *behavioral task relationships* and integration of *non-human resources* are not supported.

5.2.3 Volunteering Matters

Volunteering Matters[8] (see Fig. 5.3) is a VMS-portal located in the United Kingdom, incorporating around 30,000 volunteers each year. Within Volunteering Matters, people are able to volunteer full-time or part-time by applying for voluntary activities across the UK. Thus, Volunteering Matters primarily helps to mediate between volunteers and formal VIOs respectively help seekers.
The main feature provided by Volunteering Matters is a search for voluntary activities, which helps to apply for one of them. After entering a postal code, volunteers are able to find near volunteering opportunities, sorted by the distance to the location. After volunteers found a suitable activity, they can apply for it. Afterwards, the respective VIO will contact suitable volunteers, if they think the application was appropriate and the required skill-set of the volunteering activity is met by the volunteer.

8 https://volunteeringmatters.org.uk

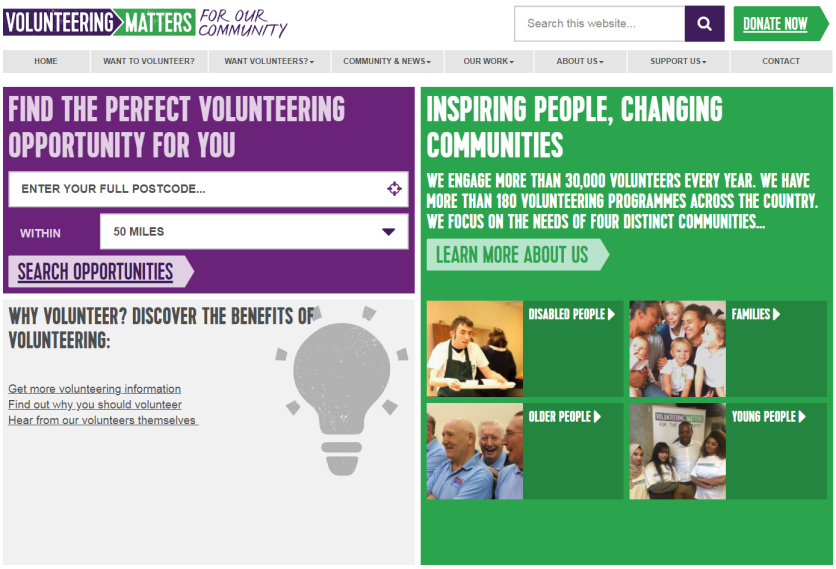


Figure 5.3: Impressions of Volunteering Matters

5.2.3.1 *Footprint Management*

Volunteering matters does not support volunteer footprints and therefore does not provide any footprint management at all.

5.2.3.2 *Organization Management*

The Volunteering Matters platform does support multiple tenants, but they are only able to state basic information about their VIO, similar to "Freiwillig".

5.2.3.3 *Social Integration*

Within Volunteering Matters, neither volunteers nor help seeker have the opportunity for social intercourse and therefore social integration is not supported. However, the web page introduces a blog to inform the volunteers as well the VIOs about their volunteering activities, which partly fulfills the needs of the social awareness management.

5.2.3.4 *Task Management*

Volunteering Matters does only allow to give a rough description for tasks (*structural task configuration*), but neither does allow for *structural* nor *behavioral task relationships*. *Task life-cycle configuration* and *non-human resource integration* are not supported and *task matching* is only viable to find voluntary work near your address, but does not incorporate other factors like preferences and competencies of the volunteer.

5.3 SUMMARY OF VOLUNTEER MANAGEMENT SYSTEMS

In our evaluation, three volunteer management systems were compared, each of which pursuing different objectives and providing functionality scopes. As a summary of this chapter, an overview about all previously stated features compared with iVolunteer (see Table 5.1) is provided. The marks in the cells indicate whether a feature is fully supported(✓), partly supported (~) or not supported(✗) by the respective VMS.

Features		Applications			
		iVolunteer	Freiwillig	Samaritan	Volunteering Matters
Footprint Management	Data Sovereignty	✓	✓	✗	✗
	Immutability	✓	✗	✗	✗
	Synchronization	✓	✗	✗	✗
	Verifiability	✓	~	✗	✗
Organization Management	Multitenancy	✓	~	✓	~
Social Integration	Communication Management	✓	✗	✗	✓
	Relationship Management	✓	✗	✗	✓
	Awareness Management	✓	✓	✗	~
Task Management	Task Properties	✓	✗	✓	~
	Task Life-Cycles	✓	✗	✓	✗
	Structural Relationships	✓	✗	✓	✗
	Behavioral Relationships	✓	✗	✗	✗
	Non-Human Resources	✓	✗	✗	✗
	Task Matching	✓	✗	✓	~

Table 5.1: Feature comparison of volunteer management systems

6 CONCLUSION AND FUTURE WORK

Contents

6.1 CONCLUSION

Conversely to existing volunteer management systems, the proposed VMS - iVolunteer - centers around volunteers, putting their requirements and needs in the foreground, including the need for more exploitation of their acquired competences and volunteering activities across various formal and informal organizations. iVolunteer allows for the integration of both, formal and informal VIOs, which are able to publish their volunteering activities in the form of tasks at the marketplaces. To realize, in contrast to most of the existing system, a decentralied VMS, a distributed architecture was developed, separating the marketplaces from each other, allowing VIOs to run their own marketplace within their own IT-infrastructure. Each marketplace is attributed with a configurable task management, including configuration possibilities for tasks to not only adapt to their internal structure with the possibility to add new properties, but also to adjust the life-cycle of tasks through the incorporation of the workflow management framework Activiti. To also allow for configuration between tasks, structural and behavioral task relationships were established, allowing to build task hierarchies in order to break down complex tasks into sub-tasks and to, e.g., handle temporal dependencies between tasks.
To manage and coordinate between the marketplaces and the users, the cross-marketplace component functions as mediator, allowing for the interoperability of marketplaces. However, this interoperability should not lead to marketplaces exchanging volunteer information between them, but on the contrary, iVolunteer establishes data sovereignty for volunteers by mitigating data silos and allowing them to store their volunteer footprint decentralized in the local repository. iVolunteer is conceptualized to give volunteers the freedom to choose which footprint entry they want to synchronize into their local repository and which footprint entries they want to publish to which marketplace. However, by giving volunteers this functionality, iVolunteer has to guarantee, that they don't exploit the system by appending invalid achievements to their local repository. Through storing hashed versions of issued

achievements in the blockchain, verification of footprint entries can be carried out, in order for organizations to be certain, that the published footprint entries are indeed valid.
In order to regulate access to the blockchain and guarantee trust into the verification, a middleware component - the trustifier - was established handling the creation of contracts, i.e., hashed footprint entries, as well as the verification of footprint entries.

6.2 FUTURE WORK

In this section, valuable extensions to the previously presented concepts will be discussed, distinguishing between structural and behavioral extensions. While the former revolves around extending the structure of shown concepts, the latter deals with extensions wrt. their internal processes.

6.2.1 Structural Extensions

While there may be countless interesting extensions originating from the domain of volunteer management systems, this section focuses on three major areas of this thesis - (i) footprint entries, (ii) task management and (iii) social management.

6.2.1.1 *Structural Extensions of Footprint Entries*

In this thesis, footprint entries included mainly earned competences and finished tasks, thus fell short and were majorly used as black box for all types of assets a volunteer may receive throughout their volunteering career. Thus, it is necessary to establish a taxonomy for all possible footprint entries (see Sect. 6.2.1.1). A first step has been made in our recent publication "(L)earning by Doing – »Blockchainifying« Life-Long Volunteer Engagement" [38] submitted to the 53th Hawaii International Conference on System Sciences. Moreover, footprint entries are not only far more diverse than displayed in this thesis, they are also subject to evolution, where, e.g., earned competences may change wrt. their level of proficiency (see Sect. 6.2.1.1). Last, in order to adequately depict competences within iVolunteer, an integration of existing competence ontologies becomes a necessity (see Sect. 6.2.1.1), not only for appropriately handling and structuring competences, but also in order to put a system in place where competences can be compared (see Sect. 6.2.3.1) respectively derived (see Sect. 6.2.3.2).

Evolutionary Aspect of Footprint Entries
In the presented model for footprint entries, the aspect of evolution was not discussed for the sake of brevity. Nevertheless, in reality, especially competences are not awarded once and exist thereforth without any change. Competences should be attributed with a level of proficiency, being ideally automatically adapted due to by volunteers carrying out tasks, that require the respective competence. Thus, by considering the feedback of how well a competence was applied throughout a task,

respectively, how often a volunteer exercises the competence, its level of proficiency would change, leading to an evolution of the competence. Furthermore, the evolutionary aspect should consider that certain competences are only valid for a certain time frame, afterwards the competence would be lost if it was not refreshed or recertified.

Footprint Entry Taxonomy
Conversely to the depicted concepts, achievements (i.e., the main footprint entries) in reality cover much more aspects than earned competences and performed tasks. A taxonomy for all different types of voluntary achievements would be needed, including, e.g., feedback form other participants (e.g., other volunteers, help-seekers), various organization-specific statuses like achieving a new position (e.g., group leader), acquiring an award for outstanding performance or honoring for their life-long participation within the organization.

Integration of existing Competence Ontology
Additionally to the necessity of developing and incorporating a taxonomy for the various footprint entries, especially competences have the be founded on existing competence models and ontologies, like the competence atlas introduced by Erpenbeck and Heyse [39], which expands on the idea of competence dimensions by allowing to combine them with each other resulting in a less rigorous model, because competences no longer have to exclusively belong to a single competence dimension, but can belong to a combination of at most two dimensions. Therefore, by combining the four dimensions with each other, 16 cross-dimensions are established. For example, competences belonging to both, the personal competence dimension and the social-communicative competence dimension include teamwork or the ability to integrate into a group. An in-depth description of the whole competence atlas can be found in [39] and [19].

6.2.2 Structural Extensions of Task Management

While the conveyed task management already represents a solid foundation wrt. structural and behavioral management, the additionally conceptualized packages, especially the resource and recommendation management were discussed briefly, only. The resource management lacks extensions for the claiming of resources for tasks and the differentiation between resources that can only be claimed once at the same time for a task and those that can be used multiple times simultaneously. Similar, recommendation management does only differentiate between recommendations and suggestions and lacks additional information about how well a recommendation would fit or feedback about why the recommendation was issued based on a volunteer's preferences or history.

6.2.2.1 *Structural Extensions of Social Management*

Last, since a detail discourse of the social management was not scope of this thesis, it depicts just a very trivial way to handle the basic objects, e.g., posting, comment, group, relationships between users, etc. Much

more effort would be needed to incorporate the basic functionality of social networks that was discussed.

6.2.3 Behavioral Extensions

Conversely to structural extensions, behavioral extensions discusses changes to the processes of the proposed functionalities. For brevity reasons, sometimes just straightforward solutions to the problems discussed in the thesis were realized, leading to less fitting solutions. In this section, possible realizations for the two depicted components - (i) matcher and (ii) profiler, as well as (iii - iv) new verification possibilities of footprint entries and the integration of (v) participants with different trust-levels will be discussed.

6.2.3.1 *Comprehensive Matching between Volunteers and Tasks*

Matching between volunteers and tasks is a traditional recommender system problem, for which numerous different approaches exist [40]. The main recommender techniques revolve around collaborative filtering and content-based recommendation as well as ontology-based recommendation. For iVolunteer, especially ontology-based recommendation jointly with the integration of a footprint entry taxonomy (see Sect. 6.2.1.1) and competence ontologies (see Sect. 6.2.1.1) would fit well, exploiting semantic dependencies within the ontologies. Nevertheless, in order to find a good fit, ontology-based recommendation should be applied in accordance with, especially, collaborative filtering, thereby not only considering ontological similarities, but also previous interaction of similar volunteers.

6.2.3.2 *Competence Derivation for Volunteers*

The profiler's main requirement revolves around deriving competences for volunteers by considering their previous competences and their volunteer activities. Similar to the realization of the matcher, machine learning approaches and semantic approaches can be applied to overcome this problem as previously stated in [41] and [42]. While a machine learning approach would learn from previously earned competences based on an annotated training set, the semantic approach would consider the competence ontology and find appropriate derivation.

6.2.3.3 *Refinement of Verification of Footprint Entries*

The current approach of verifying footprint entries revolve around hashing and comparing them with entries in the blockchain. While this guarantees the verification of footprint entries' validity, the verification of authenticity (i.e., whether the issuer of a footprint entry is authentic) and topicality, especially when considering the evolutionary aspect discussed in Sect. 6.2.1.1 are not discussed.

6.2.3.4 *Historical requests of Footprint Entries*

When considering the evolutionary aspect of footprint entries, verifying their history would give the organizations more information, e.g., when

volunteers want to publish competences to their respective marketplace. Thus not only verifying single footprint entries but also their whole history would represent a more feature-complete realization of the verification requirement.

6.2.3.5 *Trust-Levels for Participants*

Currently, every participant of iVolunteer operates on the same level of trust, e.g., volunteers which already carried out numerous tasks would be as trustworthy as newly registered volunteers. Conversely, marketplaces of large established VIOs, e.g., the Red Cross or the Fire Brigade, would be as trustworthy as informal marketplaces of, for example, neighbour assistance. In order to counteract this generalization of all participants, a system for different levels of trust for participants has to be established. Possible approaches to differentiate between different levels of trust would be the introduction of diverse types of user registration, e.g., registration via email vs registration via official document like a passport. Marketplaces could be differentiated between VIOs registered with or without their registration number.

BIBLIOGRAPHY

[1] L. M. Salamon and W. Sokolowski, *The Size and Composition of the European Third Sector*, pp. 49–94. Springer International Publishing, 2018. ISBN: 978-3-319-71473-8.

[2] M. V. MERRILL, "Global Trends and the Challenges for Volunteering," *International Journal of Volunteer Administration*, vol. 24.1, pp. 6–14, 2006.

[3] L. M. Salamon and W. Sokolowski, *Beyond Nonprofits: In Search of the Third Sector*, pp. 7–48. Springer International Publishing, 2018. ISBN: 978-3-319-71473-8.

[4] United Nations Volunteers, "State of the World's Volunteerism Report." `https://www.unv.org/sites/default/files/UNV_SWVR_2018_English_WEB.pdf`, 2018. Accessed: 2019-07-08.

[5] Salamon, Lester M. and Sokolowski, S. Wojciech and Haddock, Megan A. and Tice, Helen S. , *The state of global civil society and volunteering*. Johns Hopkins University Center for Civil Society Studies, 2013. ISBN: 1-886333-63-7.

[6] E. Kapsammer, E. Kimmerstorfer, B. Pröll, W. Retschitzegger, W. Schwinger, J. Schönböck, N. Dürk, G. Rossi, and S. Gordillo, "iVOLUNTEER: A Digital Ecosystem for Life-long Volunteering," in *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '17, (New York, NY, USA), pp. 366–372, ACM, 2017.

[7] J. Schönböck, M. Raab, J. Altmann, E. Kapsammer, A. Kusel, B. Pröll, W. Retschitzegger, and W. Schwinger, "A Survey on Volunteer Management Systems," in *2016 49th Hawaii International Conference on System Sciences (HICSS)* (T. X. Bui and R. H. S. Jr., eds.), pp. 767–776, IEEE Computer Society, 2016. ISBN: 978-0-7695-5670-3.

[8] J. Cravens and R. Jackson, "Survey of software tools used to track and manage volunteer information." `http://www.coyotecommunications.com/tech/volmanagesoftware.pdf`, 2012. Accessed: 2019-07-08.

[9] M. Raab, "A prototypical approach for a competency-based task allocation system in the context of voluntary organizations," Master's thesis, University of Applied Science Upper Austria Hagenberg, 2016.

[10] N. Mundbrod, F. Beuter, and M. Reichert, "Supporting Knowledge-Intensive Processes through Integrated Task Lifecycle Support," in *2015 IEEE 19th International Enterprise Distributed Object Computing Conference*, pp. 19–28, Sep. 2015.

[11] N. Mundbrod and M. Reichert, "Configurable and Executable Task Structures Supporting Knowledge-intensive Processes," in *36th International Conference on Conceptual Modelling (ER 2017)*, no. 10650 in LNCS, pp. 388–402, Springer, November 2017.

[12] N. Mundbrod and M. Reichert, "Flexible Task Management Support for Knowledge-Intensive Processes," in *21st IEEE Int'l Enterprise Distributed Object Computing Conference (EDOC 2017)*, pp. 95–102, IEEE, October 2017.

[13] G. Kappel, S. Rausch-Schott, and W. Retschitzegger, "Coordination in Workflow Management Systems - A Rule-Based Approach," in *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents [ASIAN 1996 Workshop]*, pp. 99–120, Springer-Verlag, 1998.

[14] Vézina, M and Crompton, S, "Volunteering in Canada." `https://www150.statcan.gc.ca/n1/en/pub/11-008-x/2012001/article/11638-eng.pdf?st=sGRvYBqt`, 2012. Accessed: 2019-07-08.

[15] S. Boudebza, F. Azouaou, and O. Nouali, "Ontology-Based Approach for Temporal Semantic Modelling of Social Networks," in *Proceedings of the 2015 3rd International Conference on Future Internet of Things and Cloud*, FICLOUD '15, pp. 736–741, IEEE Computer Society, 2015.

[16] H. Waheed, M. Anjum, M. Rehman, and A. Khawaja, "Investigation of user behavior on social networking sites," *PLOS ONE*, vol. 12, pp. 1–19, 02 2017.

[17] M. Vacura and V. Svátek, "Ontological analysis of human relations for semantically consistent transformations of FOAF data," *CEUR Workshop Proceedings*, vol. 631, pp. 15–27, 2010.

[18] G. Cheetham and G. Chivers, "The reflective (and competent) practitioner: a model of professional competence which seeks to harmonise the reflective practitioner and competence-based approaches," *Journal of European Industrial Training*, vol. 22, pp. 267–276, oct 1998.

[19] J. Erpenbeck and W. Sauter, *So werden wir lernen!: Kompetenzentwicklung in einer Welt fühlender Computer, kluger Wolken und sinnsuchender Netze*. Springer Berlin Heidelberg, 2013. ISBN: 9783642371806.

[20] V. Heyse and J. Erpenbeck, *Kompetenztraining: 64 Informations- und Trainingsprogramme*. Schäffer-Poeschel, 2004. ISBN: 9783791022635.

[21] M. T. Strebler, D. Robinson, and P. Heron, *Getting the Best Out of Your Competencies (IES Report 334)*. Institute for Employment Studies, 1997. ISBN: 1-85184-260-8.

[22] T. Hoffmann, "The meanings of competency," *Journal of European Industrial Training*, vol. 23, no. 6, pp. 275–285, 1999.

[23] F. Delamare Le Deist and J. Winterton, "What Is Competence?," *Human Resource Development International*, vol. 8, no. 1, pp. 27–46, 2005.

[24] Object Management Group, "Business Process Model and Notation." `https://www.omg.org/spec/BPMN/2.0/PDF`, 2011. Accessed: 2019-07-08.

[25] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd ed., 2015. ISBN: 978-1-4899-7637-6.

[26] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." `https://bitcoin.org/bitcoin.pdf`, 2008. Accessed: 2019-07-08.

[27] E. Kapsammer, B. Pröll, W. Retschitzegger, W. Schwinger, M. Weißenbek, and J. Schönböck, "The blockchain muddle: A bird's-eye view on blockchain surveys," in *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*, iiWAS2018, pp. 370–374, ACM, 2018.

[28] P. Fraga-Lamas and T. M. Fernández-Caramés, "A Review on Blockchain Technologies for an Advanced and Cyber-Resilient Automotive Industry," *IEEE Access*, vol. 7, pp. 17578–17598, 2019.

[29] K. Francisco and D. Swanson, "The Supply Chain Has No Clothes: Technology Adoption of Blockchain for Supply Chain Transparency," *Logistics*, vol. 2, no. 1, p. 2, 2018.

[30] S. Mann, V. Potdar, R. S. Gajavilli, and A. Chandan, "Blockchain Technology for Supply Chain Traceability, Transparency and Data Provenance," in *Proceedings of the 2018 International Conference on Blockchain Technology and Application*, ICBTA 2018, (New York, NY, USA), pp. 22–26, ACM, 2018.

[31] E. E. Bessa and J. S. B. MARTINS, "A Blockchain-based Educational Record Repository," in *ADVANCE 2019 - International Workshop on ADVANCEs in ICT Infrastructures and Services*, vol. 1 of *ADVANCE 2019 - International Workshop on ADVANCEs in ICT Infrastructures and Services*, pp. 1–11, Jan. 2019.

[32] J. Gresch, "An Educational Blockchain for the University of Zurich (UZHBC)," Master's thesis, University of Zurich, 2018.

[33] C. Brunner., F. Knirsch., and D. Engel., "SPROOF: A Platform for Issuing and Verifying Documents in a Public Blockchain," in *Proceedings of the 5th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pp. 15–25, INSTICC, SciTePress, 2019.

[34] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pp. 30:1–30:15, ACM, 2018. ISBN: 978-1-4503-5584-1.

[35] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform." `https://github.com/ethereum/wiki/wiki/White-Paper`, 2014. Accessed: 2019-07-08.

[36] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs*. O'Reilly Media, Inc., 2013. ISBN: 978-1449358068.

[37] J. Schönböck, J. Altmann, E. Kapsammer, E. Kimmerstorfer, B. Pröll, W. Retschitzegger, and W. Schwinger, "A Semantic MatchMaking Framework for Volunteering MarketPlaces," in *Trends and Advances in Information Systems and Technologies*, pp. 701–711, Springer International Publishing, 2018. ISBN: 978-3-319-77703-0.

[38] E. Kapsammer, B. Pröll, W. Retschitzegger, W. Schwinger, M. Weißenbek, and J. Schönböck, "(L)earning by Doing – »Blockchainifying« Life-Long Volunteer Engagement," 2019. Submitted to 53th Hawaii International Conference on System Sciences (HICSS).

[39] V. Heyse and J. Erpenbeck, *Kompetenzmanagement: Methoden, Vorgehen, KODE® und KODE®X im Praxistest*. Waxmann Verlag GmbH, 2007. ISBN: 9783830968252.

[40] J. K. Tarus, Z. Niu, and G. Mustafa, "Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning," *Artificial Intelligence Review*, vol. 50, pp. 21–48, Jun 2018.

[41] J. Martinez-Gil, A. L. Paoletti, and K.-D. Schewe, "A smart approach for matching, learning and querying information from the human resources domain," in *New Trends in Databases and Information Systems*, pp. 157–167, Springer International Publishing, 2016.

[42] M. Tomassen, "Exploring the Black Box of Machine Learning in Human Resource Management : An HR Perspective on the Consequences for HR professionals," August 2016.