This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 870811

SPICE

Social cohesion, Participation, and Inclusion through Cultural Engagement

D3.3 Final User and Community Models

Deliverable information	
WP	WP3
Document dissemination level	PU Public
Deliverable type	DEM Demonstrator, pilot, prototype
Lead beneficiary	UH
Contributors	UCM
Date	28/04/2022
Document status	Final
Document version	1.0

INTENTIONALLY BLANK PAGE

Project information

Project start date: 1st of May 2020

Project Duration: 36 months

Project website: https://spice-h2020.eu

Project contacts

Project Coordinator

Silvio Peroni

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Department of Classical Philology and Italian Studies – FICLIT

E-mail: silvio.peroni@unibo.it

Project Scientific coordinator

Aldo Gangemi

Institute for Cognitive Sciences and Technologies of the Italian National Research Council

E-mail: aldo.gangemi@cnr.it

Project Manager

Adriana Dascultu

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Executive Support Services

E-mail: adriana.dascultu@unibo.it

SPICE consortium

No.	Short name	Institution name	Country
1	UNIBO	ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA	Italy
2	AALTO	AALTO KORKEAKOULUSAATIO SR	Finland
3	DMH	DESIGNMUSEON SAATIO - STIFTELSEN FOR DESIGNMUSEET SR	Finland
4	AAU	AALBORG UNIVERSITET	Denmark
5	OU	THE OPEN UNIVERSITY	United Kingdom
6	IMMA	IRISH MUSEUM OF MODERN ART COMPANY	Ireland
7	GVAM	GVAM GUIAS INTERACTIVAS SL	Spain
8	PG	PADAONE GAMES SL	Spain
9	UCM	UNIVERSIDAD COMPLUTENSE DE MADRID	Spain
10	UNITO	UNIVERSITA DEGLI STUDI DI TORINO	Italy
11	FTM	FONDAZIONE TORINO MUSEI	Italy
12	CELI	MAIZE SRL	Italy
13	UH	UNIVERSITY OF HAIFA	Israel
14	CNR	CONSIGLIO NAZIONALE DELLE RICERCHE	Italy

Executive summary

D 3.3 provides a report on the development and testing of the final versions of the user and community modelling components in SPICE. To date, after continuous study of the user modelling requirements of the different case studies, the initial user model and community model data structures were refined and finalized, as well as APIs for accessing the data. In addition to the mechanism for explicitly (manually) setting and defining user modelling data (user characteristics) that was developed and demonstrated to case studies users, a reasoning mechanism was developed and integrated into the user modelling component. An initial mechanism for community modelling was implemented and experimented in simulation. A challenge we faced (and probably continue to face) is the uncertainty about specific requirements of case studies and the unstable situations in the museum case studies. Some are due to the COVID-19 issues that could eventually lead to some readjustments of the original case studies while others may be simply the result of the evolution of the case studies. This led us originally to develop flexible mechanisms that accommodate diverse requirements. Still, it may be needed to further adapt the user model and user modelling component during the 3rd year. The details of the technology are described in this deliverable. In this stage the specific internal reasoning mechanisms of the user and community modelling were further developed and adapted according to emerging requirements of the case studies. We also spent a major effort integrating the different technologies both within WP3 and between the other packages.

Document History

Version	Release date	Summary of changes	Author(s) -Institution
V0.1	20/03/2022	First draft released, WP3 review	Belen Diaz Agudo (UCM), Iris Reinhartrz-Berger, Tsvi Kuflik and Alan Wecker (UH), Guillermo Diaz
V0.2	21/04/2022	Final draft for external review	
V1.0	28/04/2022	Final version submitted to REA	UNIBO

Table of Contents

1. Introduction

WP3 Delivery 3.3 focusses on providing the technological infrastructure that enables reasoning about individuals, their characteristics and preferences, the explicit communities they identify themselves with and the implicit communities that evolve from analysing content contributed by these individuals. WP3, itself, is composed of 4 closely related components (see figure 1): (1) individual and community models (existing models in figure 1), which are the data structures that contain information about individuals and communities (concepts taken from WP6 ontologies) and stored in the linked-data hub (included in D3.3); (2) user modeller (circled in yellow in figure 1) and community modeller (circled in green in figure 1) which are the reasoning mechanisms that monitor the users continuously, reason about their behaviour and infer their preferences and community relatedness and update the models accordingly (included in D3.3); (3) textual content analysis (D3.4) that provides input data (taken from the user interface and analysed) for the user and community modelling components to reason about; and (4) a recommender system (D3.6) that uses the user models and **scripts** (guidelines/instructions for activities, generated by WP6) for guiding the process of content recommendation to users.

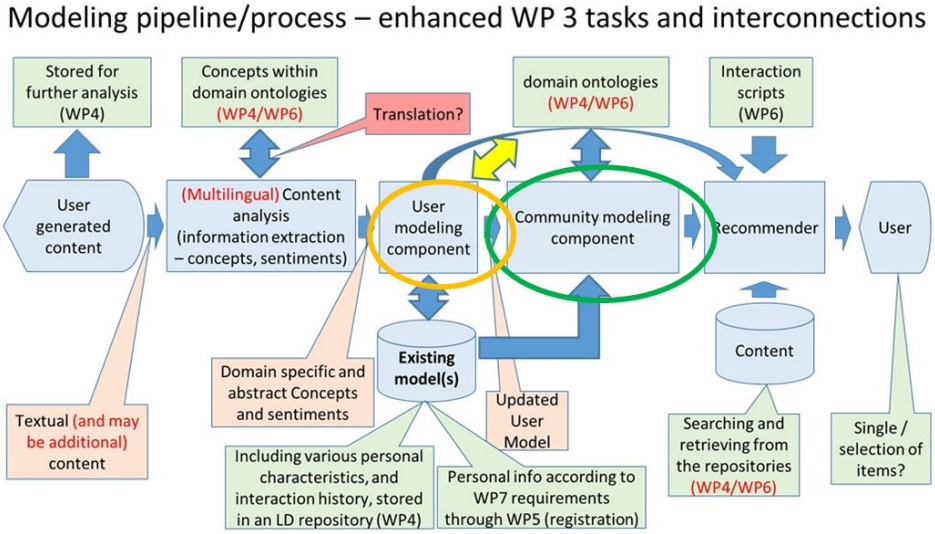


Figure 1: The user modeller and the community modeller and the internal and external interaction within WP3 and of WP3 with other WPs. The user modeller is circled in yellow; the community modeller is circled in green. The user and community models are stored in the LDH and the modellers continuously reason and update them. The analysed user generated content is used as an input and the user and community models are used by the recommender.

D3.3 focusses on the individual and community user models and user modelling components. The document is organized as follows:

It starts with an abstract description of the user model and the user modeller and its services, and then describes the community model and community modeller and its services. Following is a description of the internal and external links of WP3. Finally, there are conclusions reached so far. An appendix includes detailed APIs, usage examples and code of the different services.

2. User Model

2.1 Motivation and justification

In the SPICE project, **user models** represent the individuals that are interacting with the system. They are key elements (together with the community models) used to guide the process of content recommendations to individuals, taking into consideration individual and community interests, as well as script guidelines (WP6), to search and identify relevant users' contributions, to provide alternative interpretations of objects,

to promote the social contagion among users and to emphasize the similarities and differences within and across communities.

In this document we review the studies carried out to identify what user characteristics are relevant (and needed to be represented) for the project. The information gathered through a series of project meetings guided the development of the initial version of the user model (a data structure containing users' information) as well as a communication protocol to access the data and a user modelling component to maintain it. The initial versions were refined during the 2nd year as new requirements were made by the case studies. The procedure was as follows: first the case studies submitted scenarios, these were examined for user and community characteristics which were categorized within known generic user model categories. The results of these items were then discussed in meetings with each of the case studies.

With respect to the user model, deliverable D3.1 described the interim model for SPICE and the current deliverable describes the final version developed as the result of ongoing interactions with case studies leaders and other work package leaders.

2.2 General Structure

The deliverable consists of prototype code of REST APIs[1], built using the SPRING boot framework[2] and this document which describes the role which the user model plays in the overall project. In terms of deployment in the project the intention is that each case study would instantiate its own version of the REST server. In addition, a REACT frontend[3] was developed which gives an example of how to use the REST services.

The model is constructed in such a way that use cases can either call them in batch mode or continuously. The CM can be notified via an API when there is a change (again batch or continuously).

The User Model (UM) component provides five types of major services (controllers). The first concerns configuring the user model. The second concerns the creation, retrieval, update, deletion (CRUD) of users within the model. The third concerns the properties of an individual user and provides CRUD services for each property. Each case study can configure the user model to use the properties it feels necessary for its scenario. The values of the user model can be grouped to form communities implicitly in the community model.

There are a number of basic concepts:

User – This is an individual who is modelled based on **properties** that are organized into different **categories** (Identity, Demographics, Traits, Beliefs/Values, Interests, Skills, Communities, Current Contexts). A property is configured by giving it a name, how it is constrained (what the allowable values are) and an aggregation strategy (how we handle multiple calls to configure this property). Aggregation strategies can be: latest (last one given), first (first value given all others ignored) average (mean of all values given) or weighted average (mean with later entries given more weight). In addition, when a property is added one can add information such as in what **context** the information was added, what the **source** of the information was and whether it was **explicitly** given by the user or **implicitly** derived based on some observed behaviour or other factors.

Next is a table that presents the different categories followed by a further explanation of the categories and their possible use in the case study scenarios:

[1] https://en.wikipedia.org/wiki/Representational_state_transfer
[2] https://spring.io/projects/spring-boot
[3] https://reactjs.org/

Table 1: User model information categories

Category	Stability	Examples	Structure +(derivation, date)	Derivation mode (derived from)	Scenario (Case Studies, possible properties)
Identity	High	id#, password	type, name, value,	Explicit	All
Demographics	Medium-high	A18Y, religion, ethnicity	type, name, parameter,	Explicit	DMH-Age, Gender, Education, Languages, Organizations
Traits	Medium-High	Personality, Learning Style, Preferred Curation Type, Current Falk Identity	type, name, degree, parameter	Explicit	
Beliefs (Values)	Medium- High		type, name, degree, parameter	Derived	Hecht - Patriotic, Religious
Interests	Medium	Abstract concepts,	type, name, value on scale, {concept, activity}	Explicit (questionnaire) Implicit (User Activity)	All, implicit groups based on interests
Skills	Medium-High	Curation, Writing (Language) Reading (Language)			DMH Activities
Communities	Medium-High			Implicit	All
Current Context	Low	Social, Spatial, Temporal, Emotional, Environment, System			Useful for scripts

Identity – These are properties which identify the user (e.g., ID, email, password). Necessary if we want to use the models over more than one session. The identity will contain a specific property called **ExplicitGroup** which will be the name of the group which the user visited with if he came with as a group.

Demographics – Descriptive of the user which are fairly stable (e.g., age, gender, place of birth). These can be used to help form explicit communities (see below).

Traits – Values which describe the user (e.g., personality, learning style). These can be used as shortcuts to determining properties.

Beliefs/Values - Items the user holds (thinks) and are important/to be of value. These can be useful in the formation of implicit groups and/or common ground.

Interests - Items that the user likes. These could be evidenced by how long s/he views an artefact connected to an interest. These can be used to improve user satisfaction or find common ground.

Skills – Things that the user is good at or believes s/he is good at. Useful for determining what scripts to run

Communities - Either explicit communities obtained from the user info or implicitly derived from community model. Used by recommender for choosing content

Current Contexts – Info about the user's current environment (system, display capabilities, weather). May be useful in determining which scripts to run.
The following table shows the categories and some of their characteristics (Structure is what items make up the values and always include the source (derivation) and date added/last updated).

2.3 Accessing the User Model

The following is a list of the REST APIs based on the three major services described above.

The user controller allows for additions of users to the model. Again, it includes basic CRUD functionality, with an additional helper function which lets you get a certain property across all users. Usually, a user is created at the start of an interaction.

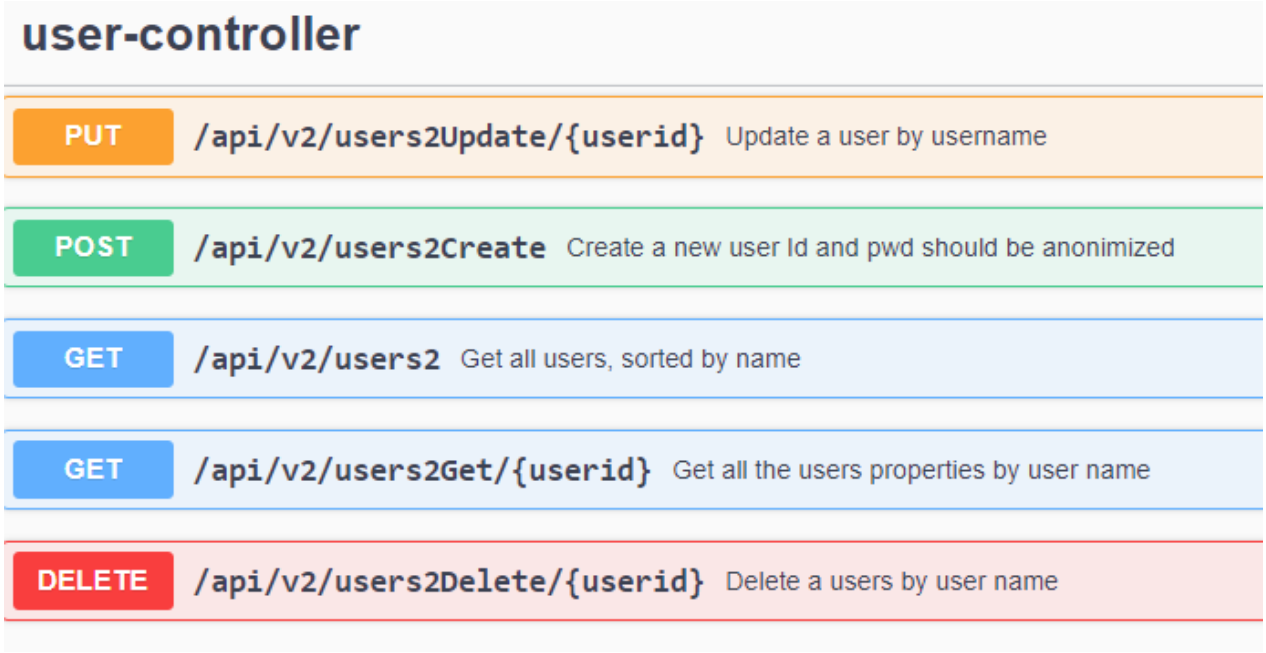


Figure 2: managing user models

The property controller allows you to manage (CRUD) each specific property for each individual user. The additional functionality includes retrieving all properties, all properties of a certain user, and all properties of a certain (property) name. Properties are either created and updated during user interactions or can be created at the end of a phase of user interactions, in anticipation of the next phase of user interactions

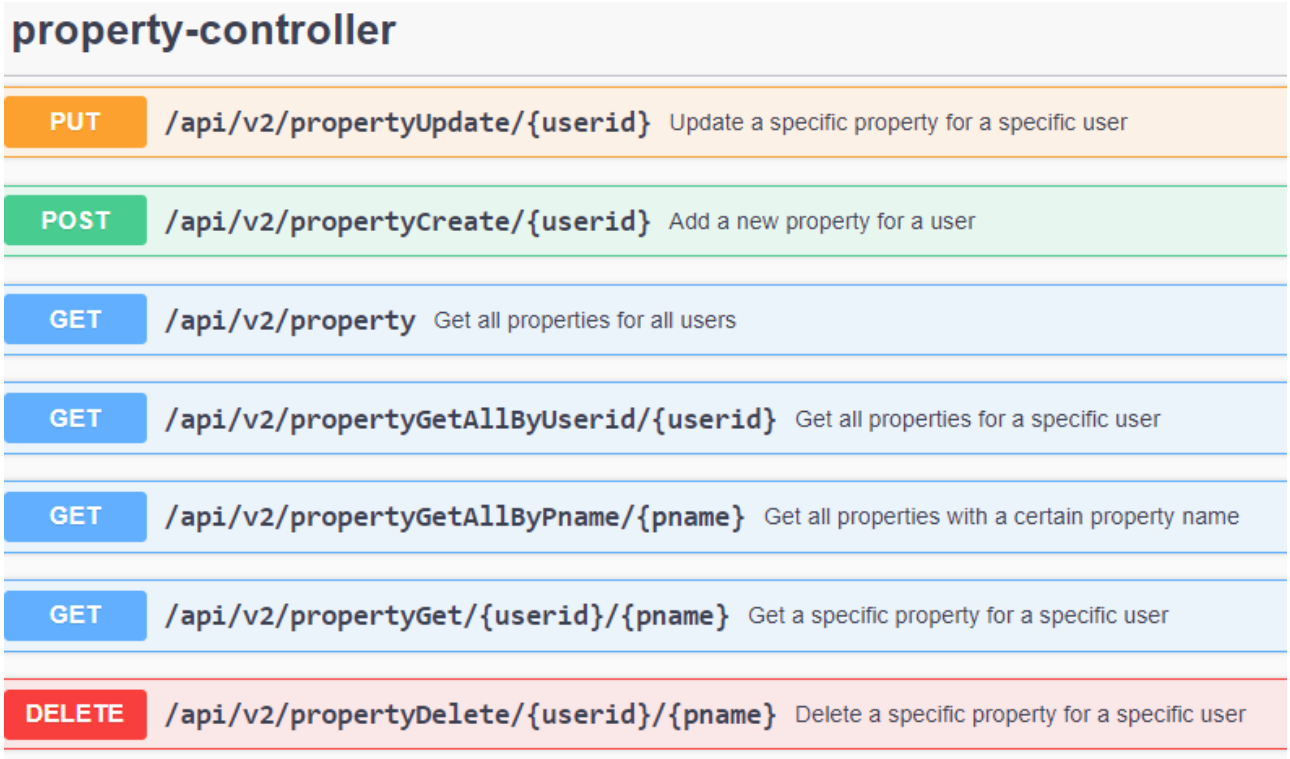


Figure 3: Property controller – an API for accessing user model (user) properties

The user-generated-content (UGC) controller allows you to manage (CRUD) each specific UGC for each individual user. The additional functionality includes retrieving all UGCs, all UGCs of a certain user, and all UGCs of a certain name. This controller is used internally by the User Model. We give UGC by the UI run it through the Semantic Analyzer and integrate results in the User Model

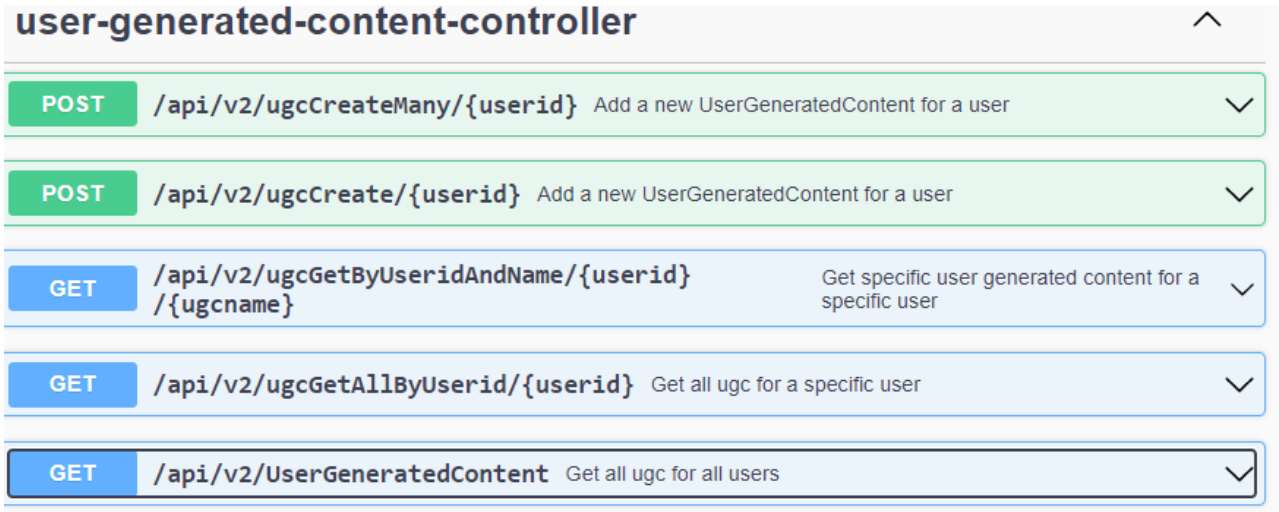


Figure 4: UGC controller – an API for accessing and managing the user generated content

See Appendix A for the details of the layout of the file structure of the code. A more detailed description of the APIs, organized by the five major services, can be found in D6.4.

2.4 Example of Use

Screenshots of the React frontend can be found in Appendix C.

We discuss here new example from a real use Case (Hecht). the hspice and studentmgr applications. The hspice mainly stores info into the user model by interactions with the students, while studentmgr can be used to query data from the UM for use by teachers and researchers. The studentmgr is based on Usermodel demo but contextualizes and groups the information into a form more usable by teachers and researchers. The hspice app is an app used both in the classroom and museum where the student responds to questions and creates content. Since the app is in Hebrew, we only provide a link to them (see:

https://hspice.haifa.ac.il/hspice

https://hspice.haifa.ac.il/studentmgr

Since React doesn't know how to make REST calls directly, we use the axios[4] library to wrap the calls. In D6.4 we give detailed information concerning the fives services that are implemented to cover the User Model API (examples of the service calls from React to the REST APIs) and used to implement the screens in React (presented in Appendix C).

3. Community Model

In the SPICE project, *communities* are key elements to search and browse contents of interests, to identify similarities and differences across users and their contributions, to provide alternative interpretations of objects, to promote the social contagion among users and to emphasize the similarities and differences within and across communities. Detection, visualisation and explanation of communities allow for the exploration, reflective reasoning and social cohesion of the users. In previous Document Deliverable D3.1 ("Prototype User and Community Model") we reviewed the initial investigations performed in SPICE regarding the types of communities required and the information to detect and represent them. In this document D3.3 we detail the advancements in the Community model, detection and visualisation. We first review the main definitions regarding the community model.

3.1. Review of main definitions

Communities are groups of users with shared characteristics. All the citizens in the same community share certain attributes. This set of shared attributes depends heavily on the data set and the characteristics of the case study. A community identifies a group of users that are heavily connected (according to a certain similarity measure) among themselves, but sparsely connected to the rest. Users are *asserted* to belong to certain **explicit communities,** but they will be *inferred* to belong to the so-called **implicit communities**.

Implicit communities are detected by community detection algorithms using two types of attributes (see next section 3.2. Data inputs for the community model): *personal* attributes from the user model (demographics, age, gender) and *interaction attributes* related to user opinions on items or abstract concepts from the *content model*. Community detection algorithms are based on assessing similarity between users. As different communities may be formed using different sets of features, the same user can be classified in different communities within different criteria, features and combination of features. All the community detection algorithms can be configured by the museum curators to assure that each community is meaningful enough for a certain museum.

In the interaction with the use cases, we have also dealt with *persona* or profiles that are related with the concept of explicit community. A **persona** is a realistic interpretation of our end users (see also D7.3 and D7.5). It characterises the range of attributes defined by a user profile. A persona represents a *user type* that might use the system in a similar way. Creating personas will help us understand users' needs, experiences,

[4] https://github.com/axios/axios

behaviours and goals. Examples: teachers, members of a certain association, deaf people, elder people. Although we understand that conceptually an explicit community is not the same as a persona, in the community model, we make analogies between both. In the analysis of the use cases, the museum curators have identified the main personas and profiles. We have correspondingly used these profiles as the main source for the definition of the set of explicit communities. These explicit communities have been formalised in one of the ontologies of WP6 (*explicit community ontology)*. It is important to note that a community (either explicit or implicit) could integrate different personas, and within the same *persona* profile we can detect different communities that differ in user attributes (i.e., classes from different schools, from different religions).

Examples of explicit communities have been defined using the persona profiles defined in some of the project use cases (in WP7) that represent user archetypes summarising common behaviours (like teachers in the children's school visits). Other examples of **explicit communities** could be a children group from a certain school, or visitors from an association. Explicit communities are those communities characterised by a set of attributes defined explicitly by curators, according to museum interests. Explicit communities can be defined using restrictions over personal attributes, for example, elderly people are users whose age is over 65 years-old, or Catholic students are users whose religion is catholic and whose education is school. Also, curators can define explicit communities using interaction attributes, for example, Picasso lovers are users who like items whose author is Picasso, and, Catholic Students are user who consider that Josephus Flavius is a Traitor (HECHT Museum Case Study). Document[5] includes a detailed list of the explicit communities.

Communities can also be classified as **persistent,** which are those that are stable in time and can be defined in the *explicit community ontology* as part of the user model; or **temporal (also virtual) communities**, which have a temporary and dynamic character and arise with new users, new opinions, stories and/or reflections. Virtual communities are detected using the community detection algorithms and can become persistent and included in the ontology if required.

SPICE communities can be **open communities** that are those where citizens can join freely, for example, “Picaso lovers” or **closed** that are those where citizens only belong according to their features (Elder people, Italian citizen).

It is worth noting that in the literature we have found other classifications of types of communities. For example, it is common to distinguish between *Communities of Practice (CoP) and Communities of Interests (CoI)* [Cantador 2011]. CoP are groups of people who get involved in a process of collective work in a shared domain of human endeavour. Members engage in joint activities and discussions, help each other, and share information. They develop a shared repertoire of resources: experiences, stories, tools, ways of addressing recurring problems—in short, a shared practice. Communities of Interest (CoI) are a particular case of CoP, which have been defined as a group of people who share a common interest or passion. They exchange ideas and thoughts about the given passion. Although members of a CoP share a common interest, we think that new CoIs will arise if we involve the interactions generated by users. This way, new CoIs will appear among members of the same and different CoPs. In SPICE, we refer to communities in general, but the community model **does not** explicitly distinguish between CoPs or CoIs as it is not required for reflective reasoning and social cohesion purposes.

The SPICE community model supports the exploration of objects and interpretations, and also helps the recommender system find contents of interest. Besides, communities of users help the recommender system avoid the cold start problem as, due to intra-community similarity, a new user can be treated like other users from the same community. The main goal of the community model is supporting the interpretation-reflection

[5] There is a document where case studies defined some explicit communities: https://liveunibo.sharepoint.com/:x:/r/sites/spice-h2020/Shared%20Documents/SPICE%20H2020%20Documents/Case%20Studies/Community%20Modelling/Explicit%20Communities.xlsx?d=wac30d24dd7014e59a93162e1337e0947&csf=1&web=1&e=akZxPY

loop and the recommendation tasks involved in this loop. More precisely, it is responsible for discovering implicit communities to reason about inter and intra relationships among explicit communities for promoting social cohesion, suggesting alternative perspectives to broaden the framework of dialogue and understanding.

The community model represents the set of all the communities (explicit and implicit) and their descriptions and relations. The community model can be queried using the API that includes endpoints with services for communities, users and contributions and explanations. The REST API documentation is available at: http://spice.fdi.ucm.es/

In the current version we (ideally) assume that the community model is always up-to-date. That is, each time a new user and/or a new contribution is included in the system, the community model has resources to recompute the whole community set.

The discussion and review of the main community detection algorithms have been described in Deliverable 3.5: Prototype clustering techniques. Community detection algorithms need to be configurable for each case study. More specifically, each use case could have a different configuration of the similarity functions from a set of predefined functions (see Catalogue of similarity measures in Annex of Deliverable 3.5). Many different alternatives have been reviewed as there is no clustering algorithm that can be universally used for every type of dataset, and there are no similarity functions that work smoothly with every clustering algorithm on every dataset. Configuration and parameter settings are crucial in the performance of a clustering algorithm.

The set of communities is dynamic and varies over time. We have already emphasised that users can belong to different communities representing different perspectives (features) of the same user. These perspectives are managed as different similarity measures. As the community model is dynamic, it needs to be updated when new information is included in the system. Some clustering algorithms have the capability to rearrange clusters when new data points are added to the dataset without running the algorithm from scratch. Visualisations techniques help users to analyse, validate and explain the clusters generated by the algorithms.

3.2 Data inputs for the Community model

According to section 3 of this document (and also Deliverable Document D3.1 - "Prototype User and Community Model"), the user models represent the individuals that are interacting with the system.

Users are described using different types of properties (demographic, cultural, skills...) contained in the user model. These properties will be called **personal attributes** for the rest of the document.

Museums store collections of items. According to Deliverable Document D4.1 (“Distributed Linked Data Infrastructure”) the linked data infrastructure supports the storage of museum collections using multiple ontologies. Users interact with items generating their own content (reflection, opinion, interpretations, comments, reaction to opinions…). Content/emotions analysis module (D3.2 - “Semantic annotation of social curatorial products”) generates information about user interests based on emotions, sentiment, attitudes, and so on (user1 likes Item1; user1 hates Item1; Item1 evokes fear on user1) and it is stored in the user model. This information links users and items. Generally speaking, these interactions can be represented as a tuple (user, item, interaction value), where the interaction value can be an emotion, a positive-negative rating, a conceptual value, etc. This information individualised for a user, will be called **interaction attributes** for the rest of the document. Note that some of the use cases have reported examples where a group of users interact with museum items as a whole (in the MNCN case study, children work in pairs and not individually). In this case, if the group members are known, these interactions will be translated into (user, item, interaction value) tuples for each member of the group. Also note that users can also generate content about abstract concepts, subjects (or beliefs) that are related to a museum activity instead of specific items from the content model (in the HECHT case study, students fill in a questionnaire about their beliefs whether Joseph Flavius is a traitor). This user generated content is also formalised as interaction attributes, represented as tuples *(user interaction activity-concept)* and stored in the user model.

As we have described in section 3.1, the Community Model distinguishes between two types of communities: *explicit communities* that are defined on demand by museum curators, and *implicit communities* that are

discovered based on user personal attributes and the information extracted from user interactions (interaction attributes).

Users are adhered to explicit communities in two ways:

- Curators adhere users to explicit communities. It is very useful when the users are anonymous (we do not know any personal attribute about the user) but activities are organized by groups of users that belong to the same community. This way, the community attributes are transferred to its members.
- Community Model adheres users to a community by assertion, checking which users satisfy the restrictions defined by the explicit community.

The later adhesion system is based on the User Model Ontology. Curators must define explicit communities based on restrictions over personal and interaction attributes that must be transformed into assertions over properties in the User Model Ontology. Figure 5 shows an excerpt of the ontology with some of the explicit communities of the case studies.

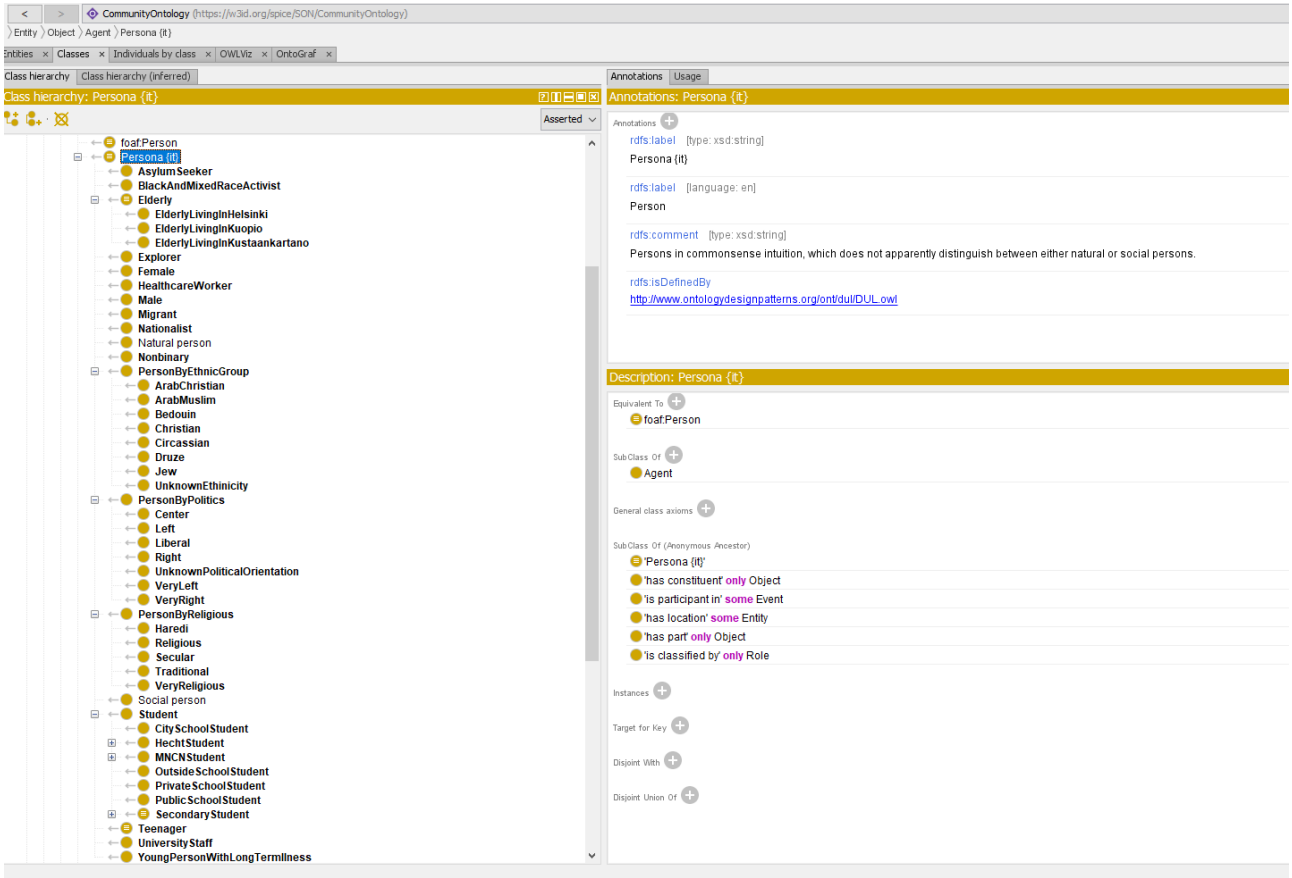


Figure 5: A snapshot of the User Model Ontology

Figure 6 shows an example of assertions.

DataPropertyAssertion(a:hasAge *a:Meg* "17"^^*xsd:integer*)	Meg is seventeen years old.
SubClassOf(DataSomeValuesFrom(*a:hasAge* DatatypeRestriction(*xsd:integer* *xsd:minInclusive* "13"^^*xsd:integer* *xsd:maxInclusive* "18"^^*xsd:integer*)) *a:Teenager*)	Objects that are older than 13 and younger than 18 (both inclusive) are teenagers.

Figure 6: Examples of assertions using a User Model Ontology

The first axiom states that a:Meg is connected by a:hasAge to the literal "17"^^xsd:integer. By the second axiom, each individual connected by a:hasAge to an integer between 13 and 18 is an instance of a:Teenager. Therefore, this ontology entails that a:Meg is an instance of a:Teenager — that is, the ontology entails the following assertion `ClassAssertion( a:Teenager a:Meg )`.

3.3 Community model relationships for reflection processes

In this section we describe an example to define the relationships that can be employed for promoting social cohesion in the context of the community model. In the figure below, users are represented as dots where the colour represents the explicit community that the user belongs to. In the example, we have two explicit communities (blue and yellow) with six users each. Implicit (discovered communities) are represented by red ellipses. They are the result of running some of the community detection algorithms described in Deliverable 3.5: Prototype clustering techniques. Before running the community model detection algorithm, it needs to be configured using similarity measures on different attributes (user attributes and/or interaction attributes) with different importance (weights). In Deliverable 3.5: Prototype clustering techniques we review different community detection algorithms based on clustering (K-means, hierarchical clustering…) [Xu 2015] and based on Graph analysis (Louvain method, modularity, Markov Clustering…) [Fortunato 2010, Yang 2016]. Both types of algorithms will heavily rely on the definition of the similarity functions. For example, we may be interested in communities of users that share similar personal profile features (demographics, age, gender) from the user model and could also combine these by identifying similar interpretations on similar contents. The use of different similarity metrics will change the set of communities in the community model, and therefore we need to include configuration capabilities using a catalogue of semantic similarity metrics.

In Figure 7, we have graphically shown that the community model is multi-layer. Every layer includes the same set of nodes (users), but each layer represents the relationship among users using a different *perspective* (i.e., similarity function): in the example, layer1 groups together users with the same evoked emotion on the same item, and layer2 uses a similarity function based on item attributes. As it is represented in the figure, the same user can be categorized into different implicit communities in different layers). On each layer, different communities have been detected using a similarity measure. Each similarity function could combine features (and weights for them) from the user model (T3.1), the interactions with artworks (interpretations) (T3.2) and content model (ontologies from WP4 and WP6).

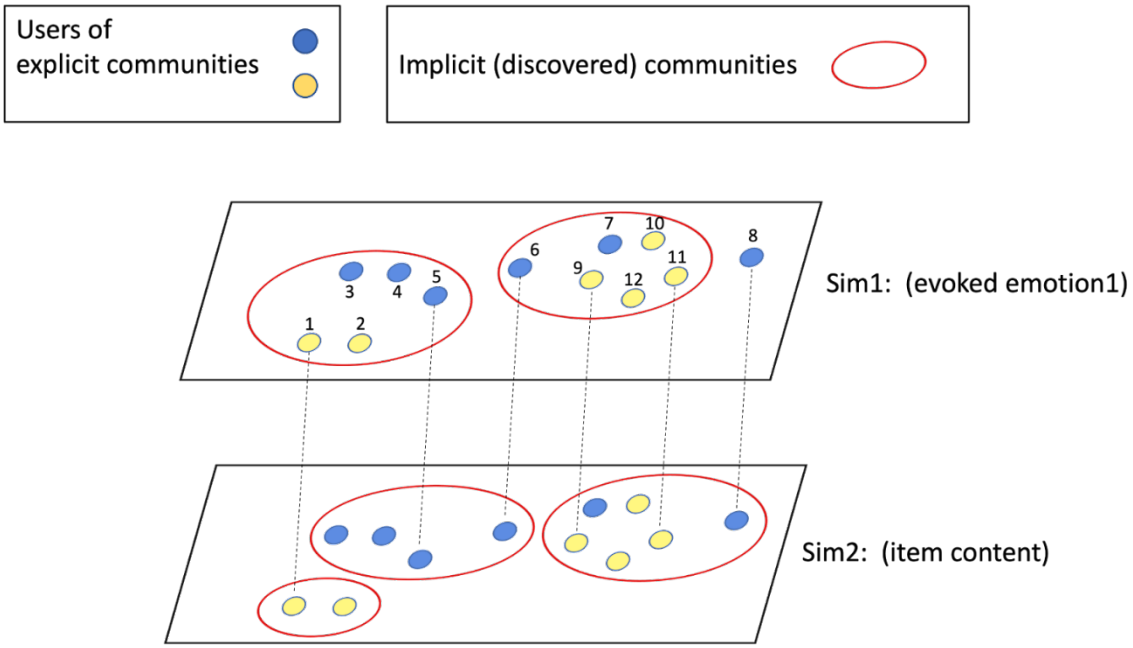


Figure 7: A multi-layer representation of the Community Model

In the following examples, we describe how reflection processes can interestingly find relationships between communities in different layers according to the users who belong to them.

Intra-community similarity

Users from the same explicit community belong to the same implicit community in the same plane/level. E.g. {3, 4, 5, 6} in Sim2

Users from the same explicit community have common implicit communities in different planes/levels E.g. {1, 2} or {3, 4, 5} in both planes/levels

Intra-community differences

Users from the same explicit community belong to different implicit communities in the same plane/level. E.g. {3, 4, 5} and {6, 7} in Sim1

Users from the same explicit community have common implicit communities in different planes/levels E.g. {1, 2} or {3, 4, 5} in both planes/levels

Inter-community similarity

Users from different explicit communities belong to the same implicit community in the same plane/level. E.g. {3, 4, 5} and {1,2} in Sim1

Inter-community differences

Users from different explicit communities belong to the different implicit communities in the same plane/level. E.g. {3, 4, 5, 6} and {1,2} in Sim2

3.4 Community model API

The Community Model API (CM-API) is the access point to the Community Model. It exposes a set of REST-based operations for accessing information about implicit and explicit communities, as well as endpoints for operations related to similar and dissimilar communities. CM-API is also employed by the User Model to notify changes in user attributes and the creation of new user generated content. The CM-API acts as a façade that hides the modules that appear in Figure 8.

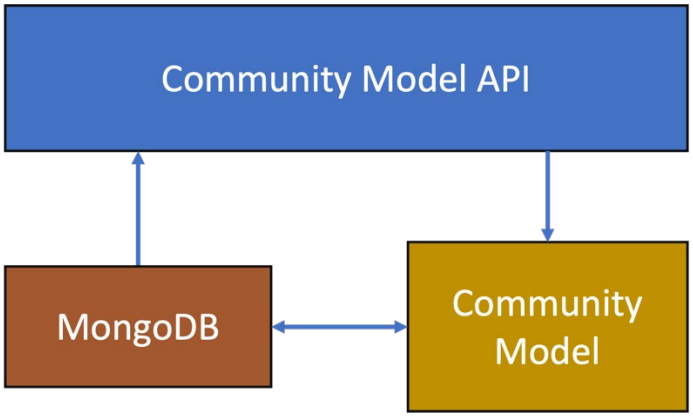


Figure 8: Overview of the CM-API infrastructure

The documentation of this API is available at http://spice.fdi.ucm.es/ and a deeper description of the CM-API is available at D6.4. APIs Specifications.

The CM-API concerns:

- Two entry points regarding USERS (Figure 9):
  - One for querying about the communities that a user belongs to.
  - One for injecting user contributions in the CM.

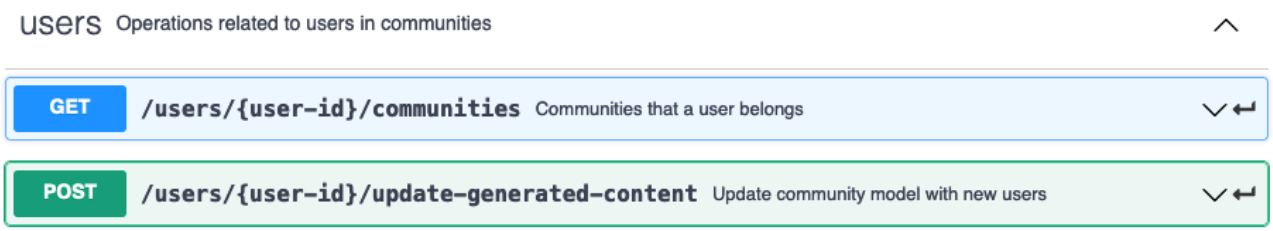


Figure 9. Entry points for user information

- Three entry points to query information about communities (Figure 10):
  - The communities within the CM.
  - The information about a concrete community.
  - The users that belong to a community.

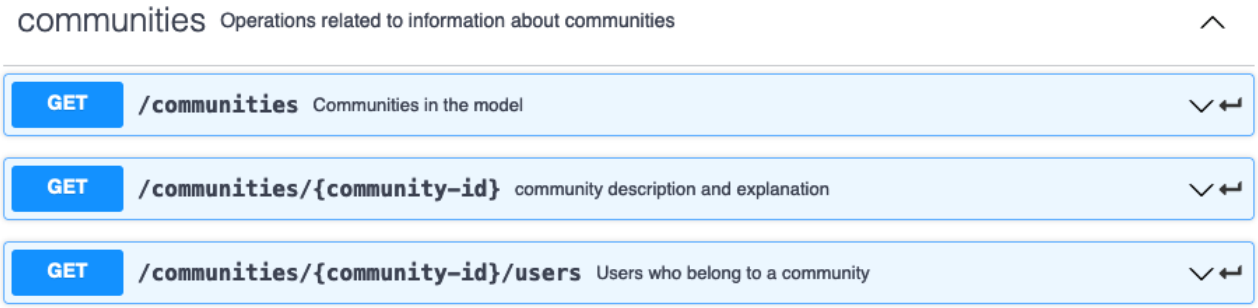


Figure 10. Entry points for information about communities

- Four entry points to provide services about similarity and dissimilarity between communities (Figure 11):
  - Two services to provide the k-most similar/dissimilar communities to a given one.
  - Two services to compute the similarity/dissimilarity between two given communities.

Figure 11. Entry points for querying similarity and dissimilarities between communities

3.5 Explanations and visualisation of the Community model

The community model includes visualisation and explanation capabilities that allow users, both citizens and curators, to understand, for example, why a certain user belongs to a certain community, what are the commonalities shared by the users of this community and what are the differences within other nearby communities. A community can be explained through the common shared characteristics, that are based on the similarity measure used to detect it. A symbolic description of these common characteristics can be built using graph-based analysis techniques (like Formal Concept Analysis). We published our results about the use of FCA to explain at [Jorro 2020].

Intra-community visualisation allows to understand the relations between explicit and implicit communities. Automatic explanations are based on common and different features shared by the community members. Figure 12 and Figure 13 show a mock interface for the visualization of these relationships and how the explanations can be displayed.

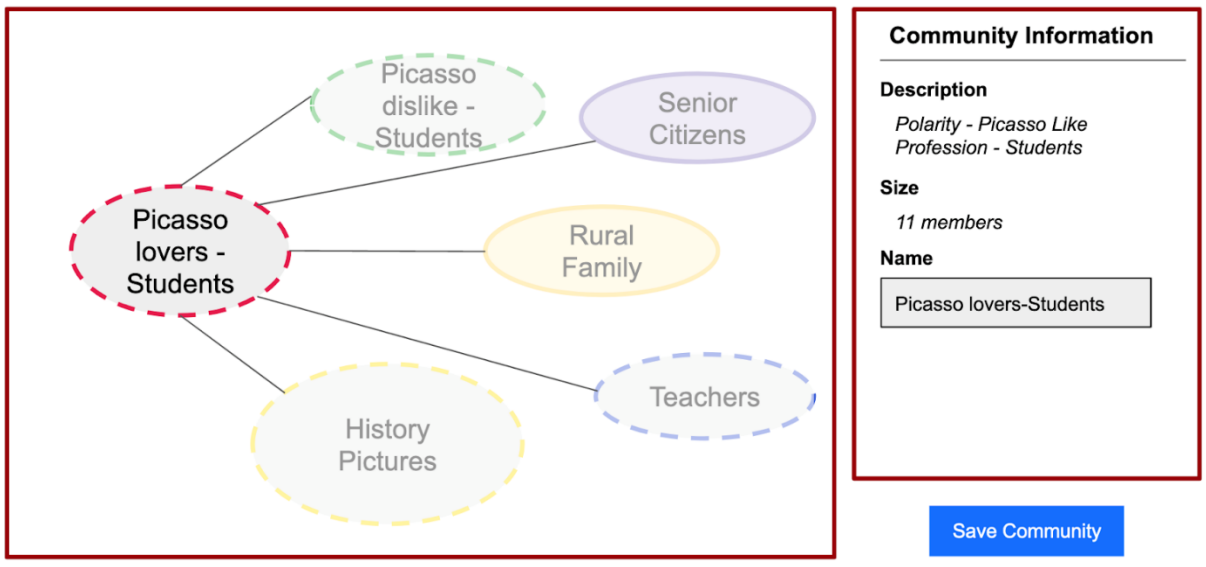


Figure 12: Visualization about the relationship between explicit (solid) and implicit (dotted) communities

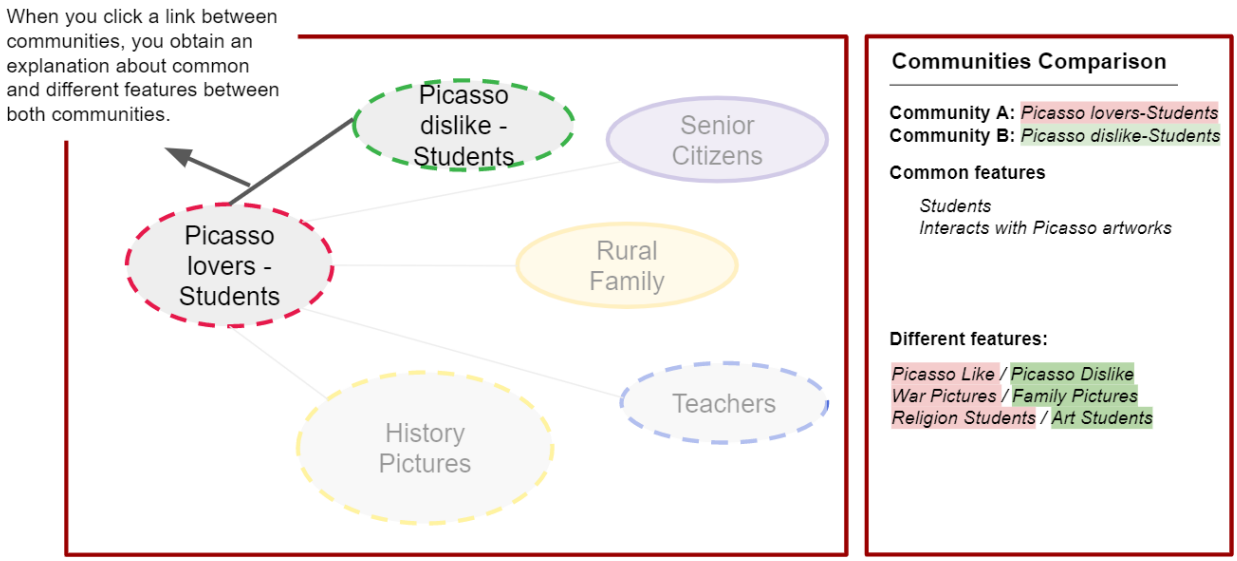


Figure 13: Automatic explanations about relationships among communities

Besides, we can visualize and get inter-community explanations based on the common properties shared by the citizens in the selected community. Figure 14 and Figure 15 shows a mock interface for the visualization of these relationships and explanations.

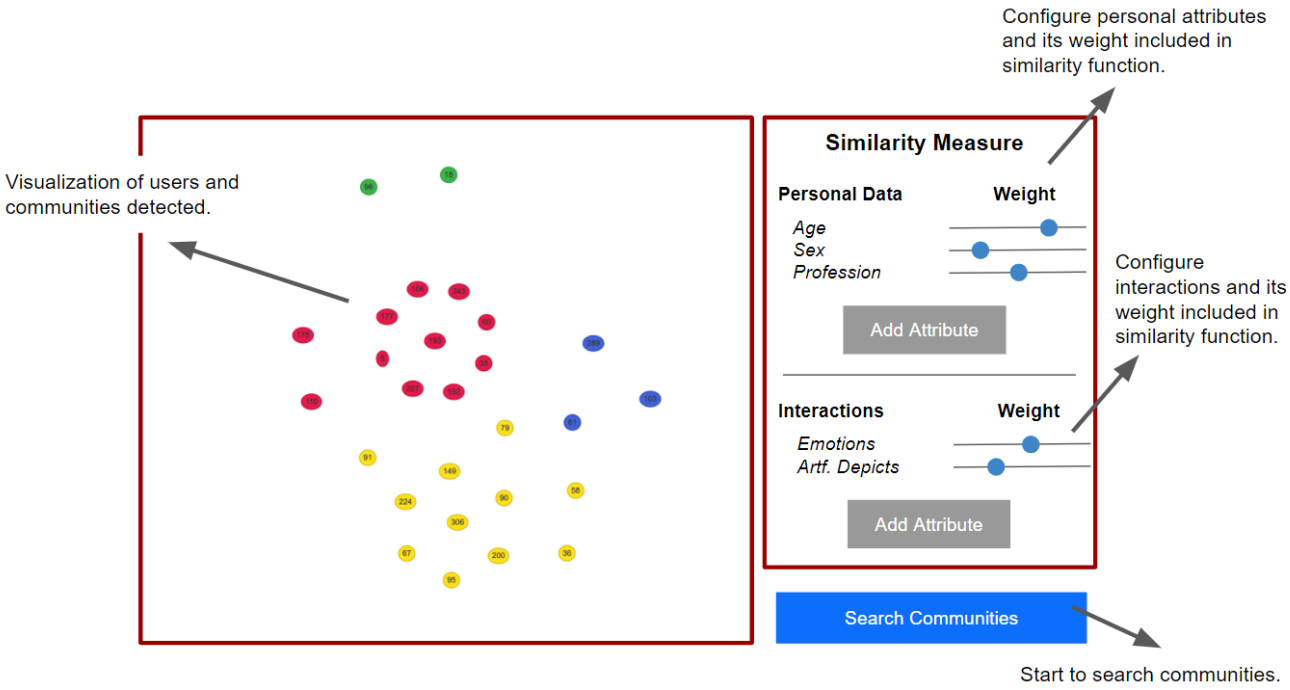


Figure 14: Visualization of inter-community similarities between citizens

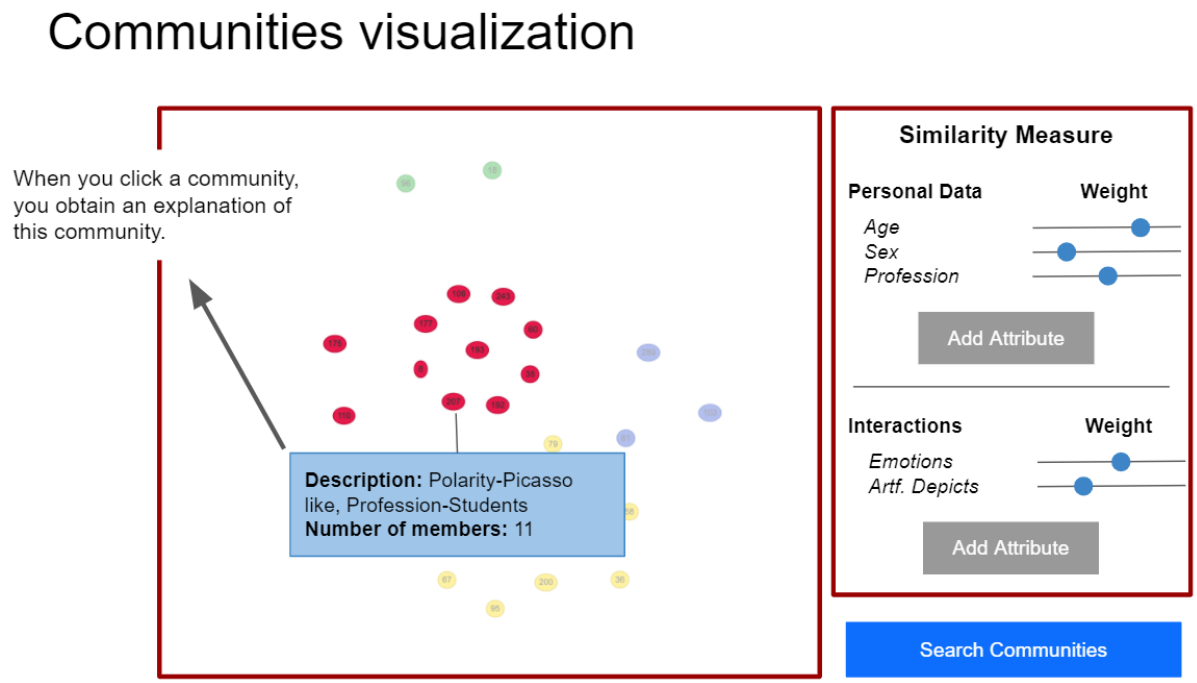


Figure 15: Explanations about inter-community relationships

When using graph analysis algorithms, the community model can be visualised as a similarity graph (network) where nodes represent users and links represent the similarity connections between them with different forces. A community identifies a group of nodes that are heavily connected among themselves, i.e., *similarity connection is strong*, but they are sparsely connected to other nodes of the graph, i.e., there is a *weak similarity connection* between them. Figure 16 shows an example, where communities are visualised through colours. Visually, the **thicker lines represent strong similarity connections** between the nodes.

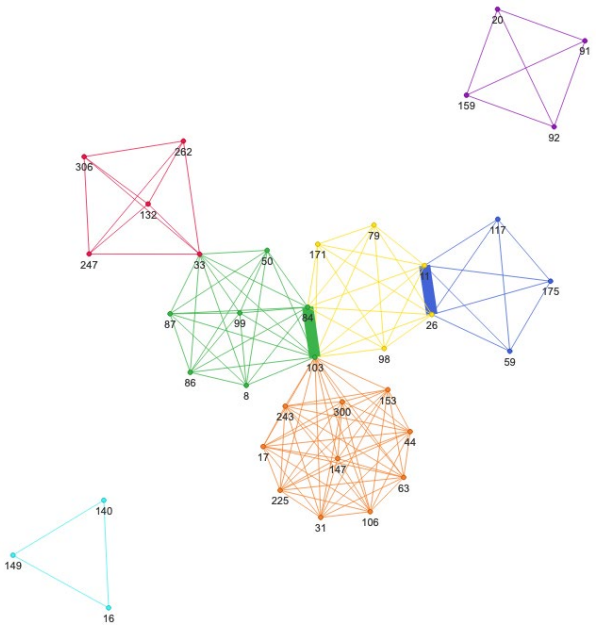


Figure 16. A graph representation of communities

The CM-API has been tested using a preliminary prototype of a visualisation tool based on graph algorithms using an example dataset of the Prado Museum, based on Wikiart Emotions dataset [Saif2018] and enhanced with synthetic data. The code of the prototype is at https://github.com/jljorro/communities_visualization.

An example of the prototype can be executed at: https://jljorro.github.io/communities_visualization/ (Figure 17). Communities can be interactively explored using this tool.

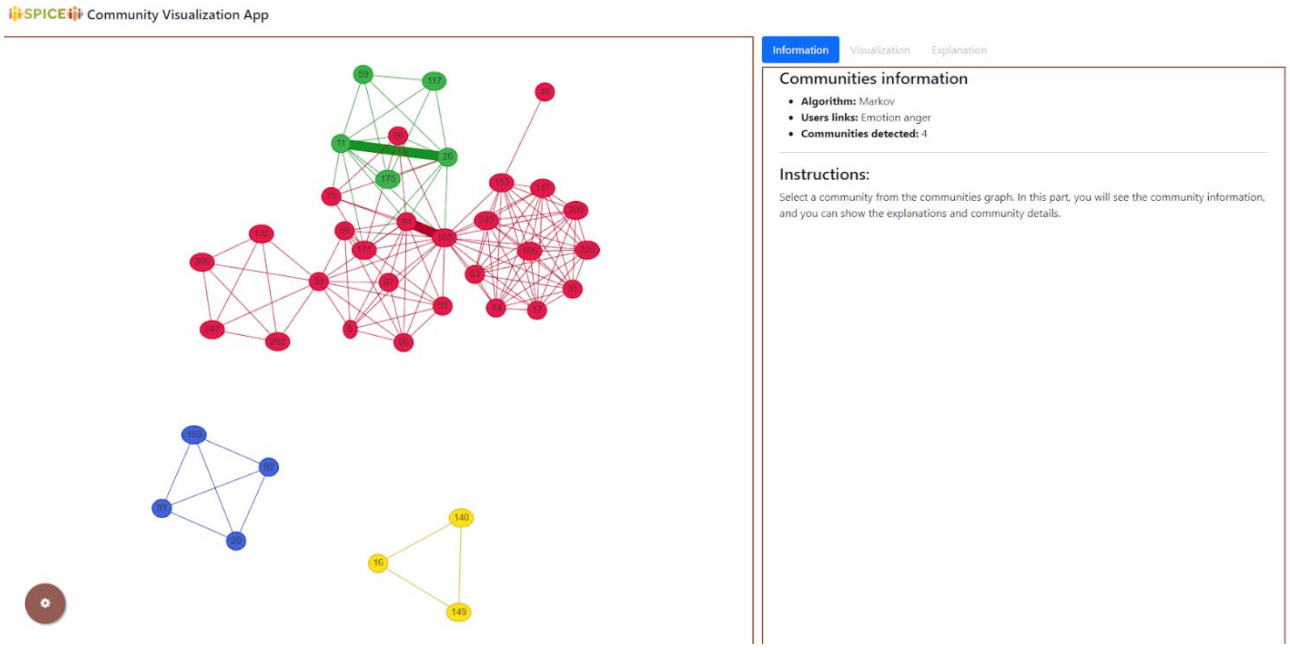


Figure 17: A snapshot of the visualization tool available at https://jljorro.github.io/communities_visualization/

In this example, two artworks are linked if they evoke the same emotion (according to Plutchik's wheel of emotions [Plutchik 2001]) for the users of the selected community. For each community we visualise the artworks that are representative of this community in the right-side graph (Figure 18).

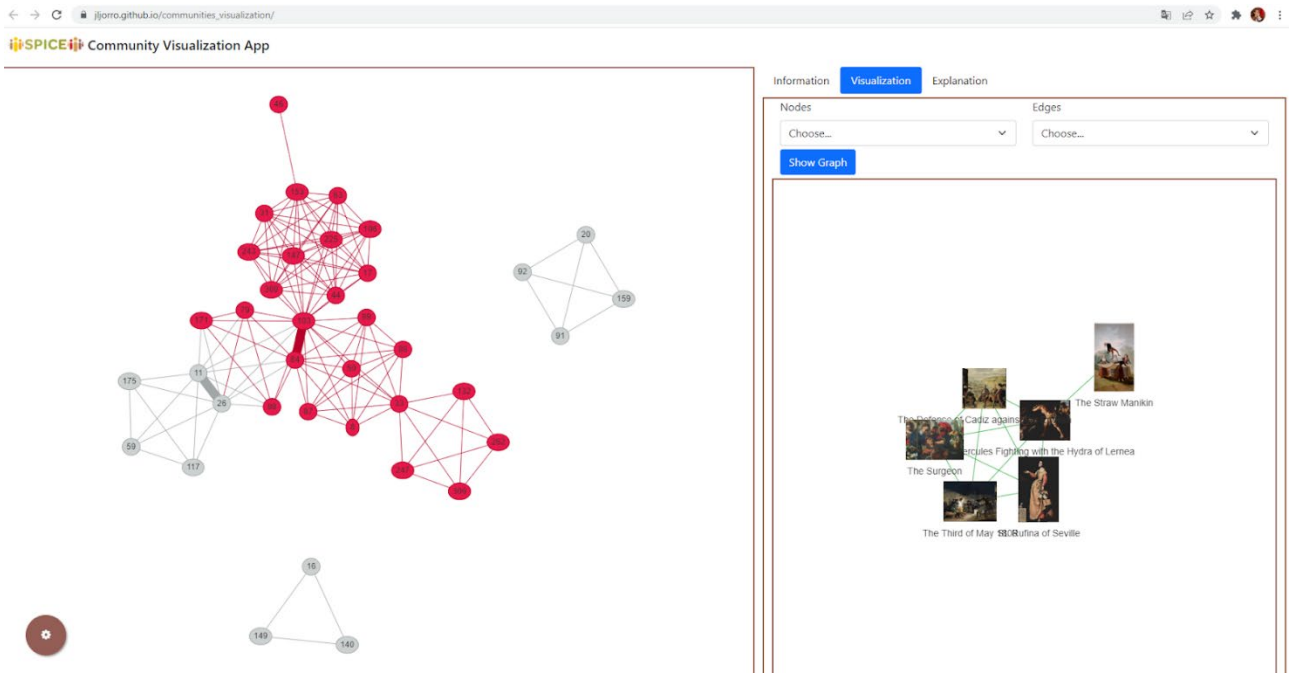


Figure 18: Community visualization and an explanation based on the representative artworks for a community.

This prototype also contains an interactive explanation (based on FCA lattices) for a given community (Figure 19).

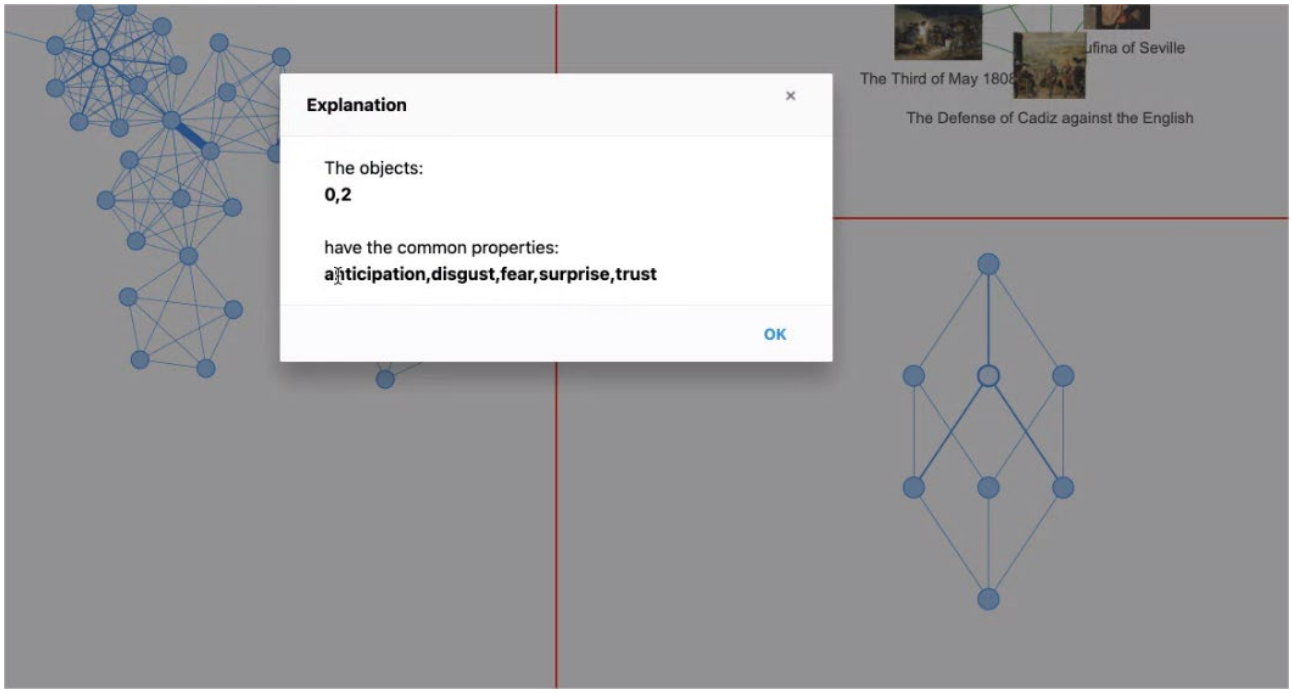


Figure 19: Explanations based in FCA lattices.

We are currently working on two types of community explanations:

- Explanations based on attributes: we have applied FCA to extract the common attributes that represent the community [Jorro 2020]. These attributes can be a combination of personal and interaction attributes.
- Explanations based on examples: the community is explained by the centroid or by a synthetically created individual that represents the statistical average or mode of the values on the real attribute values of the members of this community. The explainer individual can be also a real user whose attributes are the most representative of the community.

Figure 20 shows an example of the use of Formal Concept Analysis (FCA) to obtain intra-community explanations in the IMMA data set:

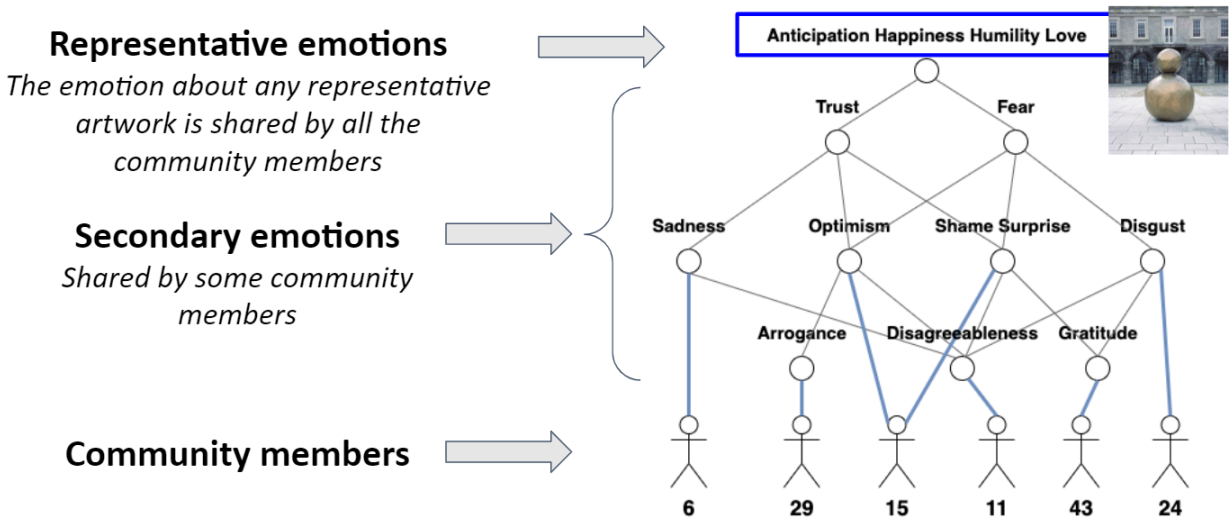


Figure 20: An example of the application of FCA for the IMMA data set

And Figure 21 shows an analogue example using the Prado Museum data set:

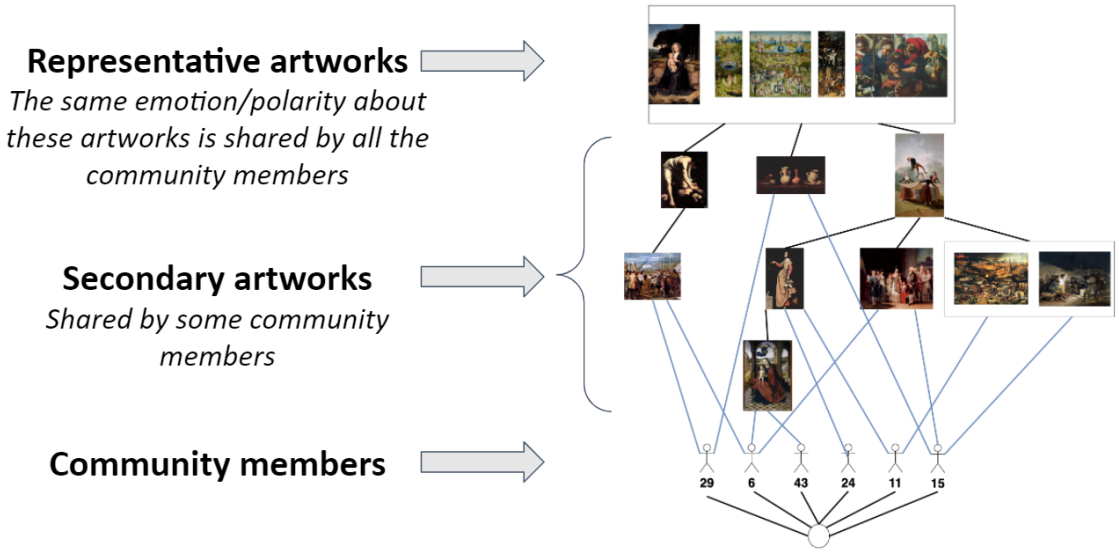


Figure 21: An example of the application of FCA for the Prado Museum data set

4. Interaction within Work Package 3

Interaction between Modules in Work Package 3 (extracted from RS Demo Hecht available at https://liveunibo.sharepoint.com/:w:/r/sites/spice-h2020/Shared%20Documents/SPICE%20H2020%20Documents/Work%20Packages/WP3/Design%20Docs/RS_Demo%20Hecht.docx?d=w55e8051aa857461393b2ea06069c37a5&csf=1&web=1&e=eOfdK9):

As noted above, WP3 is composed of several distinct components that interact and collaborate in order to provide the visitor a personalized service. Figure 24 presents a schematic interaction diagram of the overall variations of the process. The general flow of the main interaction is as follows (from top to bottom): At first, the administrator/curator configures the user and community models (UM+CM) for the case study. At a beginning of a session the visitor fills up a questionnaire for bootstrapping the user model (UM+CM). Then, during the visit, the user interacts with the system. The user may comment and/or provide input that is being analysed by the semantic annotator (SA) and in response, after the analysis of the content, the user may get recommendations for content from the social recommender (RS).

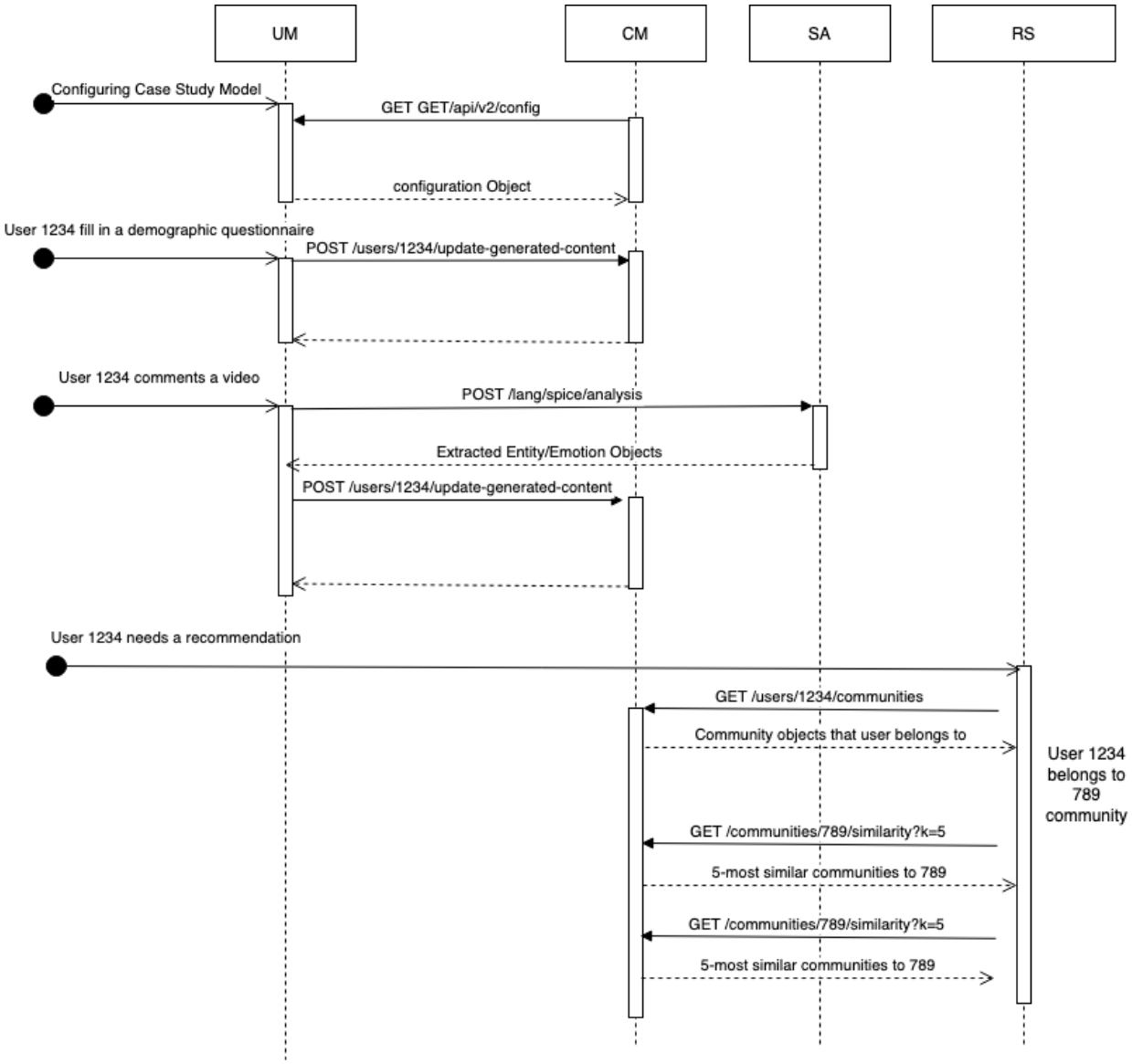


Figure 23: WP3 components interaction, from top to bottom: Case study initialization, user starts interaction and added to a community, a user interacts with the system and the interaction triggers an update to the user model and community model and finally, a recommendation is provided according to the user and community models.

When considering the role of the user model and the community model, it becomes clear that their reasoning paves the way for the social recommender to provide personalized recommendations to the current user while taking into account the users' characteristics, the communities s/he belongs to and according to the guidelines of the script/curator propose the relevant content. Figure 25 illustrates the interaction of the recommender

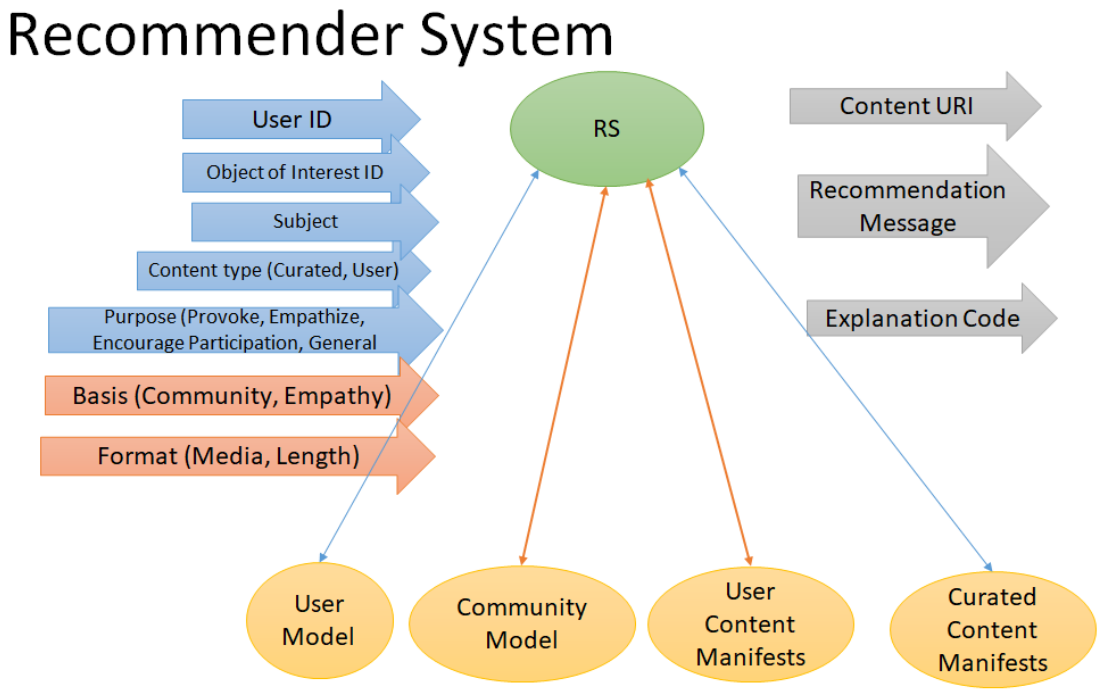


Figure 22 Recommender System Ecosystem (Parameters, Inputs, Outputs)

5. Interaction (of the UM and CM) with other Work Packages

As already noted, WP3 relies on information and guidelines provided by other WPs in order to be able to provide the required service to the visitors. The visitor interacts with her device, using the device's interface. The information from the interface is delivered, through the specific case study's application (controller). Then the user generated content is analysed and reasoned about by WP3 components, guided by the relevant script and ontologies. Then, the recommender searches for an appropriate content and delivers it back to the visitor via the user interface. Figure 26 presents a sequence diagram that illustrates

the interactions of WP3 with the other packages:

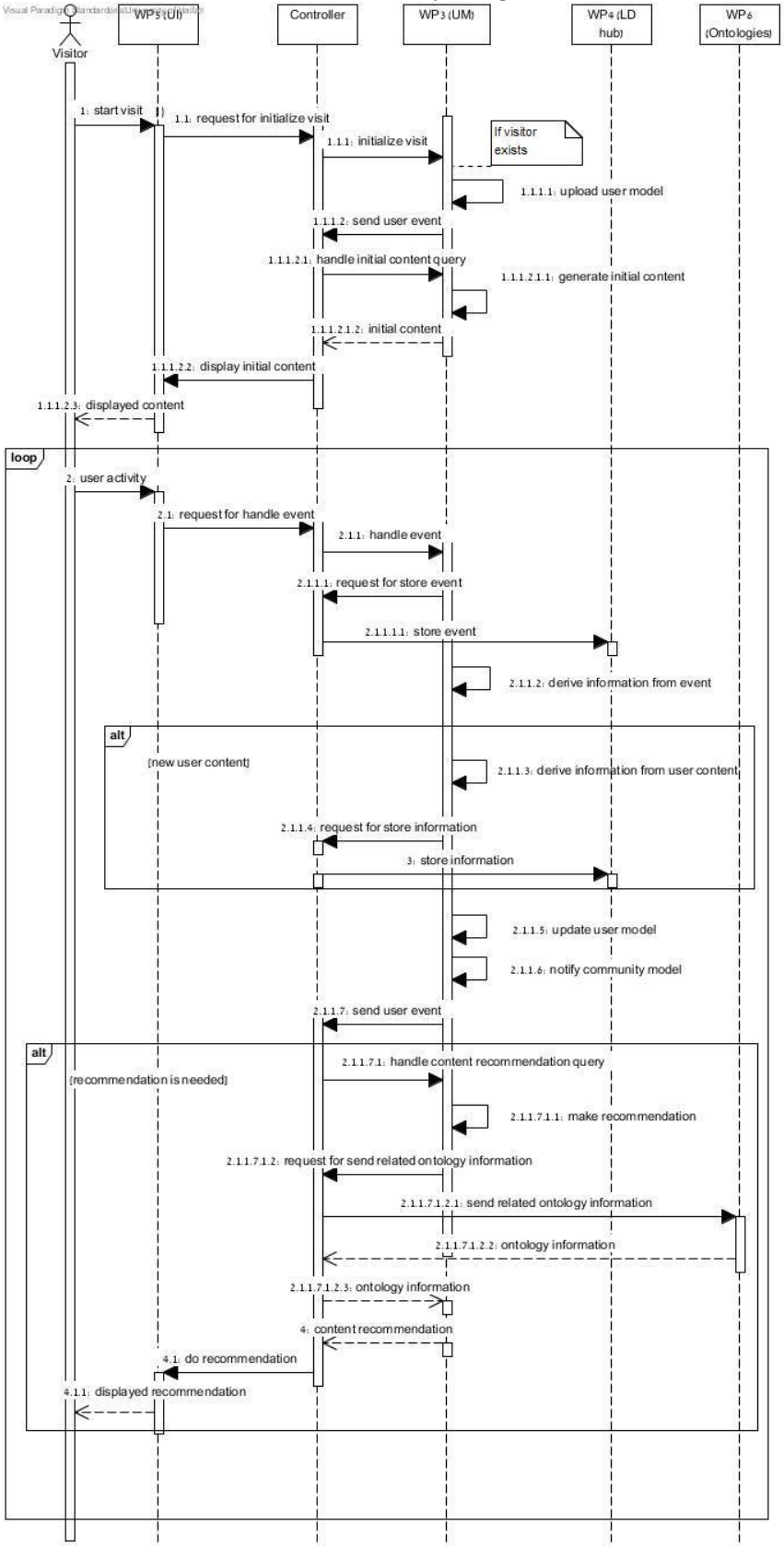


Figure 26: sequence diagram that provides a schematic interaction of WP3 with other WPs. At the top part there is a schematic description of an initialization of a user model during the beginning of a visit. Then there is a continuous interaction with the system where he interactions always start and end at the user's interface while the reasoning involves the data stored in the LDH and ontologies that guide the process.

6. Interaction with case studies

A series of meetings was held with the individual case studies during the first and second years. The motivation was to try and understand the case studies needs with respect to user and community models, in order to design them so they will be able to support these needs. Table 3 summarizes the requirements/needs of the case studies as they envision them.

Table 3: The different case studies and their envisioned needs for community models' data (of course based on the individual user data)

Case Study	Community Characteristics Explicit	Community Characteristics Implicit
Hecht	Religion, Nationality, Religiosity	Josephus, Roman Rebellion, Museum Curation
MNCN	Age, Rural vs Urban	Attitude towards climate change sustainability
GAM	Level of Physical Challenges	Emotions
DMH	Age, Gender, A18Y, Occupation, Socio-Economic Status, Location, Interests, Connection to Helsinki, Education Language	City vs Non-City Dwellers (Need to see scenario of asylum seekers)
IMMA	Visiting Groups: Blacks, LGBT+	Based on Artwork Interpretation

7. Conclusions and future work

In general, for task 3.1 the goals set for the second year were achieved. The heterogeneity of the case studies (which is a good thing) posed a challenge in terms of user modelling, posed a challenge on the development of the user and the community models and required us to suggest creative solutions to the uncertainty and lack of information about the intended use of the models in the case studies, which will be useful in ensuring that the models will be flexible and applicable to a wide range of scenarios. In both the user model and the community model components, continuous interaction with the case studies, flexible solutions to accommodate for changing requirements and simulations were adopted in order to allow us to achieve the first-year goals.

Regarding the User Model, the major (expected) challenge was uncertainty in what user characteristics may be needed for modelling users, what may be explicitly provided and what will have to be inferred. The solution was a definition of a flexible user model that may be able to accommodate any user characteristic in the form of attribute-value pair. This allows the system administrator (we hope to upgrade the demo to the level of a curator in the future) to configure the relevant user model for a specific system using a dedicated configuration tool. It also allows a combination of explicit definition of user characteristics, together with an inference mechanism that is based on concepts extracted from the user generated content and the sentiment towards them, extracted by T3.2.

This past year's tasks included:

- Developing the reasoning mechanism for updating user preferences
- Helping the case studies utilize the user/community model to enable recommendations
- Integrating the user and community model and user modelling component with the recommender system

Future tasks are:

- Integrating the user modelling mechanism into the museum case studies (WP 7.3) using Hecht as a prototype example for the other case studies
    - Integrating JSON-LD Hub

Regarding the Community Model, we have accomplished the goals associated with community representation and modelling including the study of services to communicate with other modules of the project. Until now, we have worked with synthetic user data and tested and reviewed some of the state of art clustering and community detection algorithms. Besides, we have started the tasks related with the development of tools for exploring aggregations of interpretations, visualization and interaction. We tested different clustering techniques for identifying commonalities and variabilities among the communities using artificial users and content (see Deliverable 3.5 for more details). We explored with different types of communities: explicit, implicit, persistent, virtual, temporal. Some of the use cases have already reported the explicit communities they envision in their museums.

In cooperation with WP6 and WP4 we have analysed the relationships among clusters in the community model and the content concept ontologies. This initial exploration has resulted in two lines of cooperation: the first one is the use of ontologies as knowledge to compute similarity metrics that are needed in the community detection algorithms. The second line is the relation of communities with the defined concepts from the conceptual ontologies in the project. Explanation of communities will allow to identify what are the most representative terms that will be used to link the communities with the concept ontology. This will enable users to browse the ontologies and the repository of content through the community model. This cooperation needs to be extended in the future to define semantic similarity based on ontological content, and allowing some of the implicit communities identified through the algorithms to be included as concepts in the ontologies (only for stable communities).

Various clustering techniques, like K--means or formal concept analysis have already been experimented although we need to explore further to identify the most effective techniques for the task. The homogeneous

groups of similar interpretations will be used to identify and represent the "interpretation archetypes" that will support the design of recommendation models tailored for the different user communities developed as part of task 3.1.

Finally, it is worth noting that even though the formal work on the user and community modellers/models is completed, slight changes and modifications are expected as the needs of case studies will evolve.

8. Instructions (locations of material)

This document can be found at ZENODO (DOI according to version) 10.5281/zenodo.4708753

The source code Version 2.1 for the User Model can be found at ZENODO 10.5281/zenodo.4724887. Latest version is 10.5281/zenodo.4724886.

A draft version of the user model REST API can also be found at: https://app.swaggerhub.com/apis/ajwecker/SPICE-UserModel-API/v0#/user-controller

A draft version of the community model REST API is available at: https://app.swaggerhub.com/apis-docs/gjimenezUCM/SPICE-CommunityModelAPI/v.1.1

9. References

a. User Model

Alexandridis, G., Chrysanthi, A., Tsekouras, G. E., & Caridakis, G. (2019). Personalized and content adaptive cultural heritage path recommendation: an application to the Gournia and Çatalhöyük archaeological sites. *User Modeling and User-Adapted Interaction*, *29*(1), 201–238. https://doi.org/10.1007/s11257-019-09227-6

Antoniou, A., Katifori, A., Roussou, M., Vayanou, M., Karvounis, M., Kyriakidi, M., & Pujol-Tost, L. (2016). *Capturing the Visitor Profile for a Personalized Mobile Museum Experience: an Indirect Approach*. http://chess.madgik.di.uoa.gr:10005/cvs-

Cena, F., Likavec, S., & Rapp, A. (2019). Real World User Model: Evolution of User Modeling Triggered by Advances in Wearable and Ubiquitous Computing. *Information Systems Frontiers*, *21*(5), 1085–1110. https://doi.org/10.1007/s10796-017-9818-3

Leung, R., & Law, R. (2010). A review of personality research in the tourism and hospitality context. *Journal of Travel and Tourism Marketing*, *27*(5), 439–459. https://doi.org/10.1080/10548408.2010.499058

Musto, C., Semeraro, G., Lovascio, C., De Gemmis, M., & Lops, P. (2018). A framework for holistic user modeling merging heterogeneous digital footprints. *UMAP 2018 - Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*, 97–101. https://doi.org/10.1145/3213586.3226218

Roussou, M., & Katifori, A. (2018). Flow, staging, wayfinding, personalization: Evaluating user experience with mobile museum narratives. *Multimodal Technologies and Interaction*, *2*(2). https://doi.org/10.3390/mti2020032

Spallazzo, D. (2012). *Sociality and Meaning Making in Cultural Heritage Field. Designing the Mobile Experience.*

b. Community Model

Cantador, I., Castells, P. (2011). Extracting multilayered Communities of Interest from semantic user profiles: Application to group modelling and hybrid recommendations. Computers in Human Behavior 27, 4 (July 2011), 1321–1336 https://doi.org/10.1016/j.chb.2010.07.027

Fortunato, S (2010). Community detection in graphs, Physics Reports, vol 486, Issues 3–5, 75-174, ISSN 0370-1573, https://doi.org/10.1016/j.physrep.2009.11.002.

Guo, F. Y., Shamdasani, S., Randall, B. (2011) Creating Effective Personas for Product Design: Insights from a Case Study. International Conference on Internationalization, Design and Global Development. IDGD 2011: Internationalization, Design and Global Development pp 37-46. Lecture Notes in Computer Science book series (LNCS, volume 6775) https://doi.org/10.1007/978-3-642-21660-2_5

Jorro, J. L., Caro-Martínez, M., Díaz-Agudo, B., Recio-García, J. A. (2020). A User-Centric Evaluation to Generate Case-Based Explanations Using Formal Concept Analysis. In ICCBR 2020: Case-Based Reasoning Research and Development pp 195-210 https://doi.org/10.1007/978-3-030-58342-2_13

Mohammad, S.F, Kiritchenko, S. (2018) WikiArt Emotions: An Annotated Dataset of Emotions Evoked by Art. In Proceedings of the 11th Edition of the Language Resources and Evaluation Conference (LREC-2018), Miyazaki, Japan. European Language Resources Association (ELRA) https://aclanthology.org/L18-1197.

Plutchik, R. (2001). The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. American scientist, 89(4), 344-350.

Xu, D., Tian, Y. A (2015) Comprehensive Survey of Clustering Algorithms. Ann. Data. Sci. 2, 165–193. https://doi.org/10.1007/s40745-015-0040-1

Yang, Z., Algesheimer, R. & Tessone, C. A (2016). Comparative Analysis of Community Detection Algorithms on Artificial Networks. Sci Rep 6, 30750 https://doi.org/10.1038/srep30750

Appendix

1. User Model File Structure

The structure of the file system is:

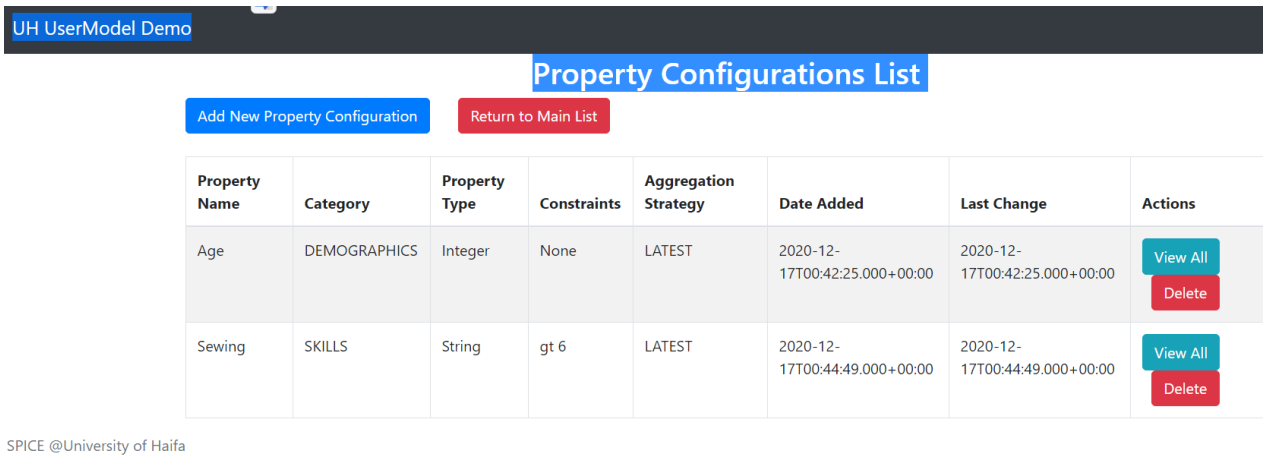
/usermodel
/usermodel/src/main/java
    il.ac.haifa.is.spice
    il.ac.haifa.is.spice.controller
    il.ac.haifa.is.spice.exception
    il.ac.haifa.is.spice.model
    il.ac.haifa.is.spice.repository
    il.ac.haifa.is.spice.security
/usermodel/src/main/resources
    /usermodel/src/main/resources/application.properties
/usermodel/src/test/java
/usermodel/doc
/usermodel/react-frontend (example frontend)
    /usermodel/react-frontend/build
    /usermodel/react-frontend/node_modules
    /usermodel/react-frontend/public
    /usermodel/react-frontend/src
    /usermodel/react-frontend/src/components
    /usermodel/react-frontend/src/services
    /usermodel/react-frontend/src/App.css
    /usermodel/react-frontend/src/App.js
    /usermodel/react-frontend/src/App.test.js
    /usermodel/react-frontend/src/index.css
    /usermodel/react-frontend/src/index.js
    /usermodel/react-frontend/src/logo.svg
    /usermodel/react-frontend/src/reportWebVitals.js
    /usermodel/react-frontend/src/setupTests.js
    /usermodel/react-frontend/debug.log
    /usermodel/react-frontend/package-lock.json
    /usermodel/react-frontend/package.json
    /usermodel/react-frontend/README.md
    /usermodel/src
/usermodel/target
/usermodel/HELP.md
/usermodel/mvnw
/usermodel/mvnw.cmd
/usermodel/pom.xml
/usermodel/usermodel-api-docs.json

1. SPICE-UserModel-API REST

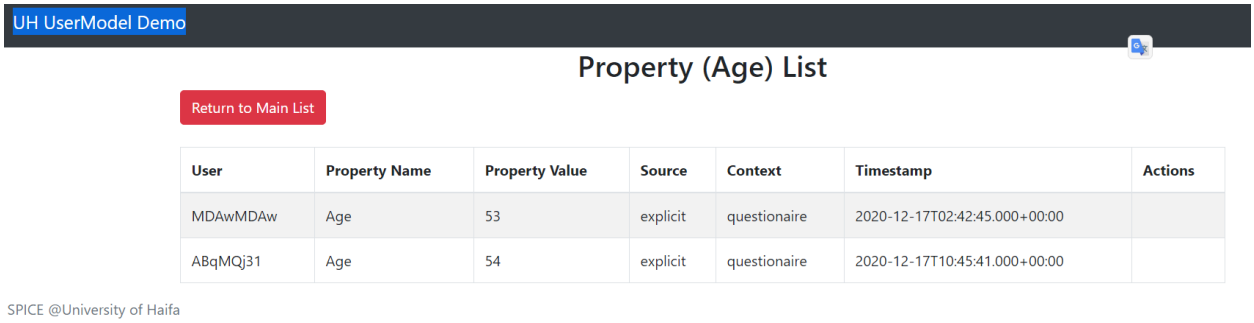
See Deliverable D6.4

1. Screenshots

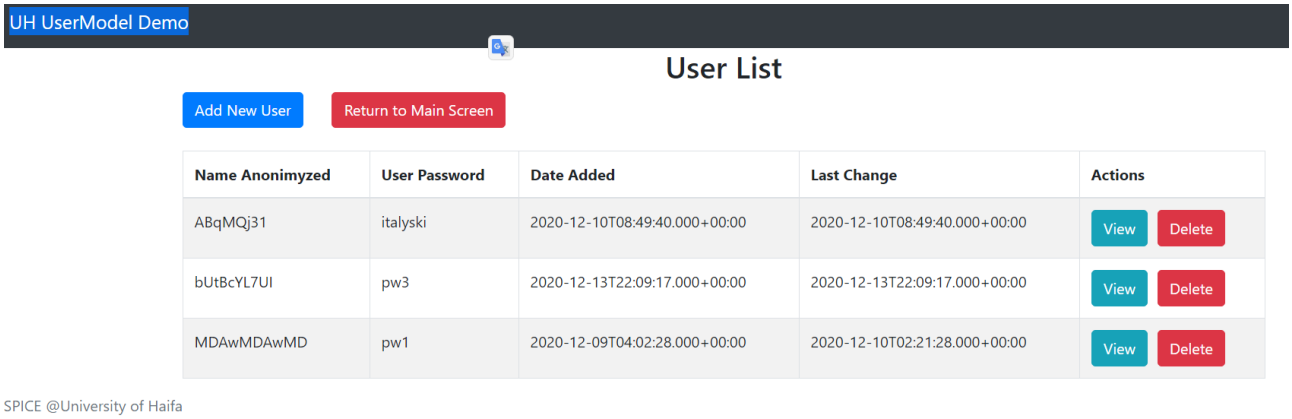
List of different properties in the configurations (From choice 1 in First screen)



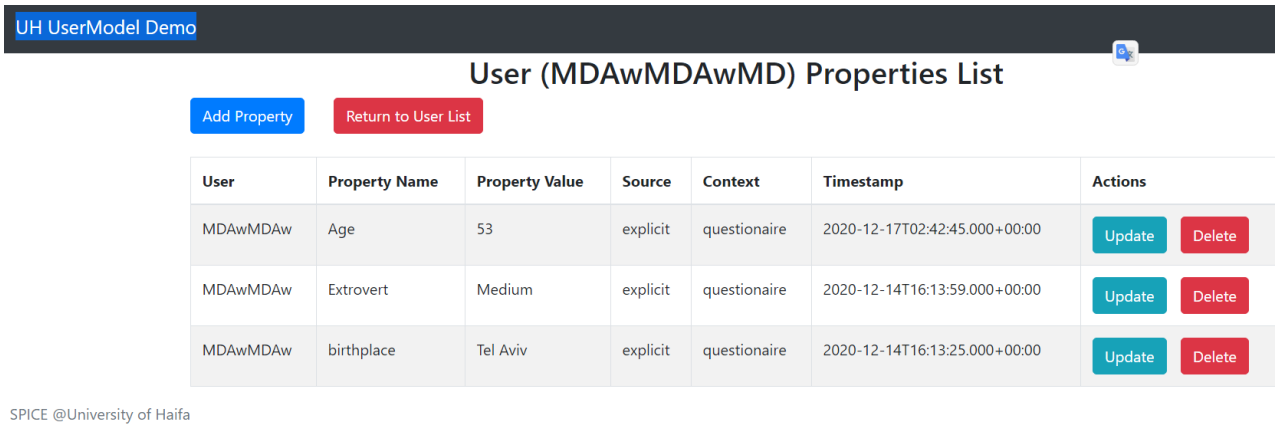
An example of all user properties with name of Age from previous screen View All



List of all users from first screen option 2



View values for a particular user from previous screen



1. React example of wrapped REST calls

1.4.1.1. User Service

```
import axios from 'axios';

const PROPERTY_API_BASE_URL = "https://hspice.haifa.ac.il/usermodel/api/v2/property";
//const PROPERTY_API_BASE_URL = "http://localhost:8080/api/v2/property";

class PropertyService {

    getPropertysByUserid(userid){
        return axios.get(PROPERTY_API_BASE_URL+'GetAllByUserid/'+userid);
    }

    getPropertysByName(pname){
        return axios.get(PROPERTY_API_BASE_URL+'GetAllByPname/'+pname);
    }
    createProperty(property, userid){
        return axios.post(PROPERTY_API_BASE_URL+'Create/'+userid, property);
```

```
    }

    getPropertyById(propertyName, userid){
        return axios.get(PROPERTY_API_BASE_URL+'Get/' + userid+'/'+propertyName);
    }

    updateProperty(property, userid){
        return axios.put(PROPERTY_API_BASE_URL +'Update/' + userid, property);
    }

    deleteProperty(userid, propertyName){
        return axios.delete(PROPERTY_API_BASE_URL+'Delete/' + userid+'/'+propertyName);
    }


}

export default new UserService()
```

1.4.1.2. Property Service

```
import axios from 'axios';
const PROPERTY_API_BASE_URL = "http://localhost:8080/api/v2/property";
class PropertyService {
    getPropertysByUserid(userid){
        return axios.get(PROPERTY_API_BASE_URL+'GetAllByUserid/'+userid);
    }
    getPropertysByName(pname){
        return axios.get(PROPERTY_API_BASE_URL+'GetAllByPname/'+pname);
    }
    createProperty(property, userid){
        return axios.post(PROPERTY_API_BASE_URL+'Create/'+userid, property);
    }
    getPropertyById(propertyName, userid){
        return axios.get(PROPERTY_API_BASE_URL+'Get/' + userid+'/'+propertyName);
    }
    updateProperty(property, userid){
        return axios.put(PROPERTY_API_BASE_URL +'Update/' + userid, property);
    }


    deleteProperty(userid, propertyName){
        return axios.delete(PROPERTY_API_BASE_URL+'Delete/' + userid+'/'+propertyName);
    }

}
export default new PropertyService()
```

1.4.1.3. User Generated Content Service

```
import axios from 'axios';

const UGC_API_BASE_URL = "https://hspice.haifa.ac.il/usermodel/api/v2/ugc";
const UGC_API_BASE_URL2 = "https://hspice.haifa.ac.il/usermodel/api/v2/UserGeneratedContent";
//const UGC_API_BASE_URL = "http://localhost:8080/api/v2/ugc";
```

```
class UGCService {

    getAllUserGeneratedContent(){
        return axios.get(UGC_API_BASE_URL2);
    }


    createUserGeneratedContent(ugc, userid){
        return axios.post(UGC_API_BASE_URL+'Create/'+userid, ugc);
    }

    createUserGeneratedContent2(ugc, userid){
        return axios.post(UGC_API_BASE_URL+'CreateMany/'+userid, ugc);
    }

    getUserGeneratedContentByUseridandName(ugcName, userid){
        return axios.get(UGC_API_BASE_URL +'GetByUseridAndName/' + userid+"/"+ugcName);
    }


    getUserGeneratedContentByUserid(userid){
        return axios.get(UGC_API_BASE_URL +'GetAllByUserid/' + userid);
    }


}

export default new UGCService()
```

1.4.1.4. User History Service

```
import axios from 'axios';

const UHISTORY_API_BASE_URL = "https://hspice.haifa.ac.il/usermodel/api/v2/uhistory";
//const UHISTORY_API_BASE_URL = "http://localhost:8080/api/v2/uhistory";

class UHistoryService {

    getUHistorysAll() {
        return axios.get(UHISTORY_API_BASE_URL);
    }

    getUHistorysByUserid(userid){
        return axios.get(UHISTORY_API_BASE_URL+'GetAllByUserid/'+userid);
    }

    getUHistorysByName(pname){
        return axios.get(UHISTORY_API_BASE_URL+'GetAllByPname/'+pname);
    }

    createUHistory(property, userid){
        return axios.post(UHISTORY_API_BASE_URL+'Create/'+userid, property);
    }
    createUHistoryMany(propertys, userid){
        return axios.post(UHISTORY_API_BASE_URL+'CreateMany/'+userid, propertys);
```

```
}
```