Incorporating Situation Awareness into Recommender Systems

Jeremias Dötterl, Ralf Bruns and Jürgen Dunkel
Hannover University of Applied Sciences and Arts, Ricklinger Stadtweg 120, 30459 Hannover, Germany

Keywords: Situation Awareness, Context Awareness, Decision Support, Recommender Systems, Semantic Web Technologies, Complex Event Processing.

Abstract: Nowadays, smartphones and sensor devices can provide a variety of information about a user's current situation. So far, many recommender systems neglect this kind of information and thus cannot provide situation-specific recommendations. Situation-aware recommender systems adapt to changes in the user's environment and therefore are able to offer recommendations that are more appropriate for the current situation. In this paper, we present a software architecture that enables situation awareness for arbitrary recommendation techniques. The proposed system considers both (semi-)static user profiles and volatile situational knowledge to obtain meaningful recommendations. Furthermore, the implementation of the architecture in a museum of natural history is presented, which uses Complex Event Processing to achieve situation awareness.

1 INTRODUCTION

Recommender systems provide users with personalized recommendations to simplify and improve their decision making. Today's smartphones and sensor devices can offer valuable information about a user's situation. Unfortunately, conventional recommender systems do not consider this kind of context information and therefore cannot compute situation-specific recommendations. This lack of situation awareness might reduce the users' satisfaction as the usefulness of recommendations is often affected by temporary factors such as the users' locations, intentions, or the current weather. When users are at work, they might not appreciate recommendations to visit the cinema to see the latest blockbuster and when it is raining, they might not appreciate recommendations for outside activities like going to the beach.

In this paper, we propose a software architecture that enhances arbitrary recommendation techniques with situation awareness. The envisioned system offers recommendations that are not only adapted to the user's personal interests but also to the current circumstances within the user's environment. The system processes situational knowledge in real-time and connects it with domain knowledge to obtain meaningful recommendations. User locations can be considered as a special type of real-time context. A real-world scenario presented in the paper will show how users' indoor location can be derived by using Complex Event Processing and iBeacon technology.

The remainder of the paper is organized as follows. Section 2 introduces recommender systems and situation awareness. In section 3, we present our architectural approach. Section 4 discusses the implementation of our approach in a museum of natural history, which demonstrates the feasibility of our proposal. Section 5 explores related work in the research area of situation-aware recommender systems. Finally, in section 6 we draw conclusions and suggest possible future work.

2 RECOMMENDER SYSTEMS AND SITUATION AWARENESS

For the computation of recommendations, Content-Based Filtering (CBF) and Collaborative Filtering (CF) constitute the two major techniques. The characteristics of these recommendation techniques are well explored (Lops et al., 2011; Su and Khoshgoftaar, 2009) and implementations are readily available (e.g., Apache Mahout (Schelter and Owen, 2012)). Because CBF and CF conceptually operate on a user-items matrix, recommender systems using these techniques are sometimes referred to as *two-dimensional* (2D) (Adomavicius and Tuzhilin, 2008). Unfortunately, 2D recommender systems do not incorporate real-time context data and therefore cannot directly consider the user's current situation (Adomavicius and Tuzhilin, 2005).

Dötterl, J., Bruns, R. and Dunkel, J.
Incorporating Situation Awareness into Recommender Systems.
DOI: 10.5220/0006385106760683
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 2*, pages 676-683
ISBN: 978-989-758-248-6

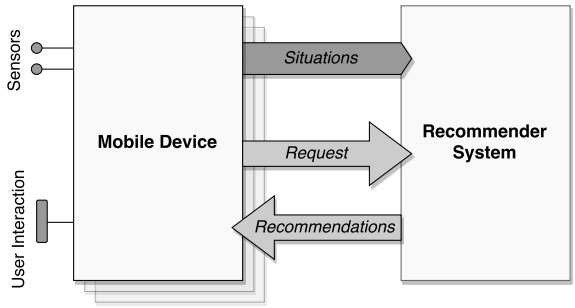

Figure 1: High-Level Architecture of a Situation-Aware Recommender System.

Situation-aware recommender systems, on the other hand, also take real-time context information into account. This way, recommendations can be provided that consider the user's current situation. To achieve situation awareness, context data has to be gathered and interpreted in real-time. Situation-related context data includes GPS data, time, temperature, calendar entries, noise, velocity, etc. For example, if GPS data shows that the user is at his work place and his phone is muted, this might indicate that he is in a meeting. Or if the user receives multiple text messages within a short time interval, he is probably engaged in a conversation with another person via text messages. A situation is typically on a higher abstraction level than the individual pieces of context it is inferred from and is characterized by a potentially short life span.

Figure 1 shows the high-level architecture of a situation-aware recommender system that acquires context data from mobile devices. In this architecture, mobile devices interpret low-level data obtained from their built-in sensors to infer high-level situations. The mobile devices provide the infrastructure, which is responsible for the detection of situations. Each mobile device sends the detected situations to the recommender system, which gathers the situations it receives from the different devices. Via the user interface of the mobile device, the user can interact with the recommender system. When the user requests a recommendation, the recommender system uses the collected situations to compute situation-aware recommendations and sends them back to the mobile device where they are displayed to the user.

While conventional recommender systems have been exploiting the strengths of CBF and CF for many years, situation-aware recommender systems cannot use CBF and CF out of the box. In this paper, we propose a technology-independent approach to incorporate situation awareness into arbitrary recommendation techniques. For instance, 2D recommender systems can be used off-the-shelf, thus allowing the use of mature and established recommendation libraries. An adaptation of the libraries' source code is not required.

3 SITUATION-AWARE ARCHITECTURE FOR RECOMMENDER SYSTEMS

Our architectural approach for situation-aware recommender systems is shown in Fig. 2. The recommender system receives a continuous stream of situations and it accepts recommendation requests at arbitrary points of time. The advent of a recommendation request triggers the following recommendation process.

1. *Situation Selection*: Situations are filtered to ensure recommendations guided by current and relevant situations.
2. *Request Refinement*: Recommendation requests are enriched with constraints and hints which are derived from the selected situations and which control the subsequent pre- and post-filtering steps.
3. *Pre-Filtering*: To the set of items available for recommendation, a filtering step is applied to discard items that are unsuitable considering the selected situations.
4. *Recommendation Computation*: A list of recommendations is computed using established recommendation techniques like CBF and CF, or alternative approaches such as Semantic Rules (Hermoso et al., 2016).
5. *Post-Filtering*: To the list of recommendations, a post processing step is applied which adjusts the list to foster recommendations that are especially well suited considering the selected situations.

To illustrate our approach, we will consider a recommendation system for a museum of natural history, which is described in some more detail in section 4.

Imagine a user called Alice who is visiting the museum. In particular, Alice is interested in reptiles and dinosaurs. Right now, it is 11:45a.m. and her usual lunch time starts at 12:00a.m., which means she is already starting to feel hungry.

She is currently located in one of the exhibition rooms and has time for visiting one more room before she intends to eat lunch in the museum's restaurant. On her mobile phone, the recommender system's client app is running, which knows the beginning of Alice's usual lunch time in the near future from her calendar. Alice requests a recommendation and the recommendation process is triggered.

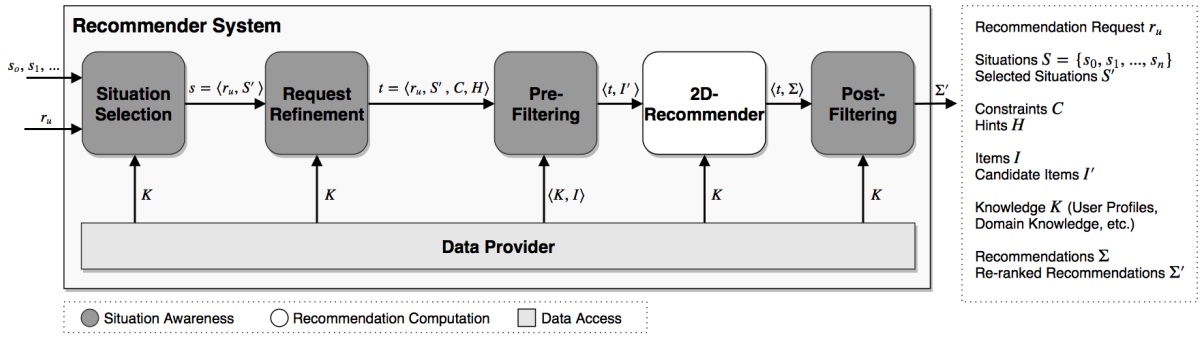

Figure 2: Architecture for Recommender Systems That Enables Situation Awareness for Conventional Recommendation Techniques.

Let $S = \{s_0, s_1, ..., s_n\}$ denote the set of situations available to the recommender system, and r_u the recommendation request issued by the the client of user u on his or her behalf. Initially, S and r_u are processed by the Situation Selection component.

3.1 Situation Selection

When a recommendation request arrives, the Situation Selection component selects the subset of situations that should influence the recommendation computation. This step is required to filter out stale situations and to limit the number of situations the recommender system has to process. Situations can become stale because they are pushed into the recommender system as soon as they are detected and the recommender system keeps them in memory until they are outdated.

Based on the input data S and r_u, the Situation Selection component creates a situation tuple $s = \langle r_u, S' \rangle$, where S' denotes the set of situations selected for r_u.

S' is constructed from S applying temporal and spatial criteria. Only situations are selected that have appeared recently and near user u or that directly concern u like his intentions or state (movement, mood, etc.).

In our scenario, only situations are selected that happened recently in rooms near Alice's location, and that concern Alice directly. This includes her current location and her intention to eat lunch soon. Tuple $s = \langle r_{Alice}, \{$ "Alice is located in room 4", "Alice's lunch time starts in 15 minutes" $, ...\} \rangle$ is forwarded to the Request Refinement component[1].

[1]In practice, the situations are encoded in a machine-readable format.

3.2 Request Refinement

The Request Refinement component accepts the situation tuple s and constructs a new tuple $t = \langle r_u, S', C, H \rangle$, where C is a set of constraints and H is a set of hints.

- Set $C = \{c_0, c_1, ..., c_j\}$ contains conditions that items *must* fulfill to be considered for recommendation. Items that do not fulfill all constraints $c \in C$ should be ignored as they are not a good match for the current state of affairs S'.
- Set $H = \{h_0, h_1, ..., h_k\}$ contains conditions that items *should* fulfill to achieve high recommendation scores. Items that match at least one hint $h \in H$ are a good match considering S' and thus should have a higher chance to be recommended to the user.

The sets C and H are directly derived from S'. The creation of C and H is controlled by domain-specific rules.

In our example, the following constraint and hint are created.

- $c_0 = \{$ "Exhibit is not located in room 3" $\}$ because room 3 is currently overcrowded and thus exhibits located in this room should not be recommended to further visitors.
- $h_0 = \{$ "Exhibit is located near the restaurant" $\}$ because visiting a room near the restaurant would be convenient for Alice as it keeps her walking distance short.

Tuple $t = \langle r_{Alice}, S', \{c_0\}, \{h_0\} \rangle$ is forwarded to the Pre-Filtering component.

3.3 Pre-Filtering

The Pre-Filtering component accepts tuple t as its input.

Via the Data Provider, the Pre-Filtering component has access to all items (here the museum exhibits) available for recommendation. The set of existing items is denoted as I. Furthermore, the Pre-Filtering component has access to the Data Provider's knowledge K, which includes user profiles (e.g. their interests, if known) and domain knowledge. Information about user interests can be obtained with explicit approaches (user interrogation) or implicit approaches (observing user behavior) (Hanani et al., 2001). The domain knowledge is provided by domain experts.

Given I, C and K, the Pre-Filtering component constructs the candidate item set I'.

$$I' = \{i \mid i \in I \wedge \forall c(c \in C \wedge i\ satisfies\ c)\}$$

Whether an item i satisfies a given $c \in C$ can be checked by consulting K.

In our scenario, all items that are not located in room 3 are members of I'. Tuple t and I' are passed to the 2D-Recommender.

3.4 2D-Recommender

The 2D-Recommender computes personalized recommendations for u using conventional recommendation techniques like CBF and CF. Only items from candidate set I' are considered.

The 2D-Recommender computes a set of recommendations, denoted as Σ. A recommendation σ is defined as a tuple $\sigma = \langle i, f \rangle$ with $i \in I'$ and $f \in \mathbb{R}$, where i is the item that is being recommended and f its assigned recommendation score expressed as a real number.

To compute recommendations for u, the recommendation techniques require the user's ID, which is contained in r_u to access the user's preferences, which are contained in the knowledge base K. Note that the 2D-Recommender does not make use of S', C, and H. Situation awareness is enabled by the pre- and post-filtering steps, but not the 2D-Recommender. This way, techniques that are not situation-aware by default, like CBF and CF, can be used without modification, which is a crucial benefit of our approach.

The 2D-Recommender passes t and the recommendations Σ to the Post-Filtering component.

3.5 Post-Filtering

The Post-Filtering component accepts t and Σ and creates a modified recommendation set Σ'. The Post-Filtering component applies two types of modifications to the set Σ to construct set Σ'.

1. The scores of items are increased according to the number of matching hints in H, as those items fit the current set of situations S' especially well. Again, whether item i satisfies $h \in H$ can be checked by consulting K.
2. If the size of set Σ, denoted as $|\Sigma|$, exceeds a certain threshold ω, the items with the lowest scores are dropped such that $|\Sigma'| \leq \omega$. This is required to limit the number of recommendations that have to be processed by subsequent components and to limit the number of recommendations that are eventually displayed to the user.

Let us assume Σ contains two recommendations $\sigma_0 = \langle$"tyrannosaur in room 2", $0.85\rangle$ and $\sigma_1 = \langle$"local aquatic birds in room 1", $0.80\rangle$. Let us further assume room 1 is located next to the museum's restaurant, whereas room 2 is not. As declared above, hint h_0 states that the recommended item should be located near the restaurant. In this scenario, σ_1 satisfies h_0, while σ_0 does not. Consequently, the recommendation score of σ_1 is increased while the score of σ_0 remains unchanged. Depending on the used algorithm, the ranking might change in favor of the local birds in room 1 (σ_1), which are conveniently located on Alice's way to the restaurant.

The newly created set Σ' is transmitted to the mobile device that requested the recommendations and is displayed to the user.

4 CASE STUDY: RECOMMENDATIONS IN A MUSEUM SCENARIO

A prototype of our system is currently developed in cooperation with the *Landesmuseum Hannover* (museum of natural history) to ensure the feasibility of our approach. Figure 3 illustrates the set-up in the museum, which provides the required technical infrastructure.

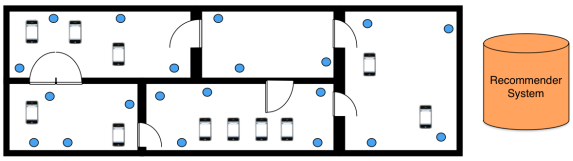

Figure 3: Technical Infrastructure in the Museum.

- *Beacons*: The dots on the museum's floor plan show the positions of the beacons, each of them marking a Point-of-Interest (POI) such as an exhibit in the museum. Beacon hardware and pro-

tocols (such as iBeacons[2] and Eddystone[3]) have recently been introduced to enable proximity services. A beacon device uses Bluetooth LE to send in a configurable frequency a unique ID that can be read by any smartphone.

- *Smartphones*: The personal smartphones of users serve as readers of the beacon signals. As soon as a user approaches a beacon within its signal range, the smartphone triggers an event carrying the unique beacon ID. These beacon events can be exploited to derive a user's indoor position. The smartphones serve as the user interface: they show information about an exhibit, which is adapted to the user's interest and capabilities (age, language skills, interests). Furthermore, they display the recommendations given by the system.
- *Recommender System*: This component is running on a server and receives user requests, user interests, and situations from the smartphones. Thereby our implementation focuses on user location as a particularly important piece of situation data. The Recommender System implements the core of the recommendation process as described in section 3.

Situation Selection can be realized with Complex Event Processing (CEP) (Luckham, 2001). CEP is a software technology to process a stream of events in real-time. A core concept of CEP is a declarative event processing language (EPL) to express patterns in the data streams: So-called event processing rules (CEP rules) are formulated for specifying event patterns that discover situations of interest.

Common EPLs contain the boolean operators $\wedge$ and $\vee$, the sequence operator $\rightarrow$, and time and length windows. The following event pattern, for example, detects all occurrences of an event of type A followed by either an event of type B or C, but only considers occurrences of the last 60 seconds: `(A → (B ∨ C))[win:time:60s]`

With these building blocks, Situation Selection rules can be expressed.

In the given scenario, Situation Selection rules work on the event model as shown in Fig. 4. The abstract type `Event` serves as a common supertype for all other types. The type `RecommendationRequest` is used for events that represent requests sent by the user's mobile device. The type `TechnicalEvent` is the abstract supertype for low-level technical events like the sensing of a beacon signal (`BeaconEvent`). The type `Situation` serves as an abstract supertype for all types that characterize concrete situations. Concrete situations like the user's location are detected by the user's mobile device and transmitted to the recommender system.

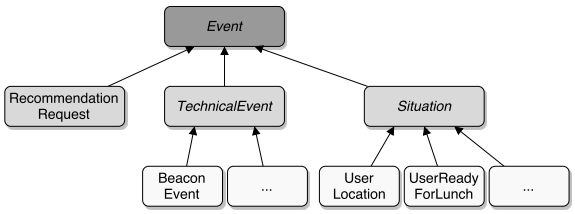

Figure 4: Event Model.

Location Inference Rule: The following CEP rule is running on the smartphones and performs a sensor fusion step and infers the new position of its user.

```
CONDITION: BeaconEvent AS b [win:time: 5 seconds]
    ∧ b.RSSI = max(RSSI)
ACTION: create UserLocation(userID, b.beaconID)
```

This rule collects all beacon events that have been read within a small time window. (Here: every 5 seconds a new window is started.) That beacon event with the maximum signal strength (RSSI) corresponds to the beacon next to the user, and a new `UserLocation` event with its `beaconID` is created. From the `beaconID`, the system can infer the room the user is currently located in.

Situation Selection Rule: The next CEP rule of the Recommender Component selects all situations of the last 5 minutes that occurred in the area of the user that issued the request.

```
CONDITION: (UserLocation AS ul
    ∧ Situation AS sit)[win:time: 5 minutes] AS s
        → RecommendationRequest AS req
    ∧ ul.userID = req.userID
    ∧ isNear(sit.location, ul.beaconId)
ACTION: forward(req, s)
```

The rule triggers when the event stream contains a `RecommendationRequest` event `req` that is preceded by a `UserLocation` event `ul` that contains the location of the user that issued the request. Furthermore, the rule selects all `Situation` events (and sub types thereof), referred to as `s`, that occurred near the user's location `ul` during the last 5 minutes. When such a pattern occurs in the event stream, the selected situations `s` are, together with the request `req`, forwarded to the Request Refinement component.

The collection of selected CEP events `s` is the set S' of our architectural concept and contains Alice's location and intention to eat lunch soon.

Request Refinement can be realized with Jena Rules[4]. Via the Data Provider, refinement rules have access to domain knowledge.

[2]https://developer.apple.com/ibeacon/

[3]https://developers.google.com/beacons/

[4]https://jena.apache.org

Domain knowledge is modeled as an OWL ontology (Hitzler et al., 2009) and contains information about the museum's different rooms and exhibits. The OWL ontology is the technical realization of the knowledge base K and the item set I.

```
:restaurant     rdf:type    :Restaurant .
:room1          rdf:type    :Room ;
                :nextTo     :restaurant .
:crocodile      rdf:type    :Item ;
                rdf:type    :Reptile ;
                :in         :room1 .
:brontosaurus   rdf:type    :Item ;
                rdf:type    :Dinosaur ;
                :in         :room2 .
:tyrannosaur    rdf:type    :Item ;
                rdf:type    :Dinosaur ;
                :in         :room3 .
```

Before entering the Request Refinement component, the selected situations are translated into RDF triples (World Wide Web Consortium, 2014), which allows the refinement rules to operate on both (rather static) domain knowledge and (volatile) situational knowledge. In RDF, the set of selected situations S' related to Alice at a certain point of time may look as follows.[5]

```
:alice  :isLocatedIn        :room4 .
:alice  :isReadyForLunch    true .
:room3  :isOccupied         true .
```

Refinement Rule 1: The following refinement rule declares that recommended items must not be located in a room that is occupied.

```
[ (?room rdf:type :Room), (?room :isOccupied true)
  -> mustNot(:in, ?room) ]
```

If a binding for the variable `?room` exists such that the two triples on the left hand side of the arrow are present in the knowledge base, the rule fires and the functor on the right hand side of the arrow is invoked. As shown above, room 3 is currently occupied. Therefore, the functor `mustNot` is invoked, which constructs a constraint object that represents the fact that items located in room 3 should be excluded from recommendation.

Refinement Rule 2: A second refinement rule checks whether the current user is ready for lunch. In this case a hint is created declaring that recommended items should be located in a room next to a restaurant.

```
[ currentUser(?u), (?u :isReadyForLunch true),
  (?room :nextTo ?res), (?res rdf:type :Restaurant)
  -> should(:in, ?room) ]
```

[5]Note that the situation 'a certain room is occupied' can be derived by an appropriate CEP rule that counts the number of users located in that room.

The invocation of the `currentUser` functor binds Alice's user ID (which is contained in the recommendation request) to the variable `?u`. Because a triple exists that states that Alice is ready for lunch and a room exists that is located next to a restaurant, the rule fires and the `should` functor is invoked. The invocation of the `should` functor creates a hint that states that recommended items should ideally be located in room 1, the only room directly located next to the museum's restaurant.

The collection of constraints and the collection of hints, which resemble the sets C and H, are passed to the *Pre-Filtering* component.

Pre-Filtering uses the constraints to select the items that will be considered for recommendation. As the items are stored in a RDF knowledge base, they can be queried with SPARQL (Harris et al., 2013). From the given constraints, a SPARQL query can algorithmically be constructed that selects all items that match the given constraints.

Pre-Filtering Query: The following query selects all items that are not located in the occupied room 3.

```
SELECT ?item WHERE {
    ?item rdf:type :Item .
    MINUS { ?item :in :room3 . }
}
```

Executed on the knowledge base, the query returns `:crocodile` and `:brontosaurus`, which are located in room 1 and 2 respectively, but not `:tyrannosaur`, which is located in room 3. Together, the selected items constitute the set $I' = \{$:crocodile, :brontosaurus$\}$.

The 2D-Recommender assigns recommendation scores to the selected items I' to obtain the recommendation set Σ. For this task, any 2D recommendation library can be used, e.g., Apache Mahout[6], which provides an implementation of CF. CF operates on item identifiers and user preferences, which are provided by the Data Provider. The 2D-Recommender explicitly does not use situations, constraints, and hints as this allows to easily exploit recommendation libraries that do not consider situational knowledge. In our scenario, the 2D-Recommender constructs $\Sigma = \{\langle$:crocodile$, 0.80\rangle, \langle$:brontosaurus$, 0.85\rangle\}$.

Post-Filtering can be realized – similarily to the Pre-Filtering step – with SPARQL. From the given hints, SPARQL queries are constructed that check which items $i \in I'$ match the given hints. For each hint an item matches, the item's score is increased.

Post-Filtering Query: The following SPARQL query selects all items that satisfy the earlier constructed hint.

[6]https://mahout.apache.org

```
SELECT ?item WHERE {
    ?item :in :room1 .
}
```

Because the museum's area about crocodile is located in room 1 (and therefore on Alice's way to the restaurant), it is selected by the query and its recommendation score is increased accordingly. For the brontosaurus, on the other hand, the hint is not satisfied and the recommendation score keeps unchanged. Therefore: $\Sigma' = \{\langle$:crocodile, $0.90\rangle, \langle$:brontosaurus, $0.85\rangle\}$

An impression about the implemented app gives Fig. 5. For a more detailed description of the presented approach, please refer to (Dötterl, 2016).

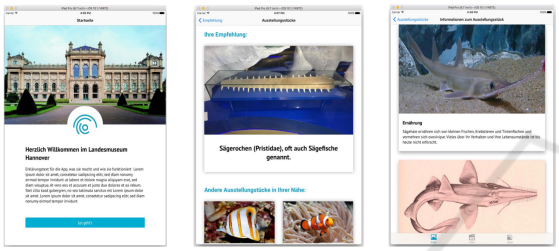

Figure 5: Museum Recommender App.

5 RELATED WORK

A large number of different approaches exist to infer situations from given context data, which include techniques like fuzzy logic, bayesian networks, or complex event processing (Renners et al., 2012; Rizou et al., 2010; Ye et al., 2012). To add situation awareness to recommender systems, often knowledge-based solutions are employed where recommendations are computed via domain-specific rules operating on ontologies (Bouzeghoub et al., 2009; Ciaramella et al., 2009; Hermoso et al., 2016). In these approaches, the two major recommendation techniques CBF and CF are generally neglected. Our architecture combines both worlds: ontology-based situation awareness and the recommendation techniques CBF and CF are merged into a single solution.

There also exist different proposals that aim to incorporate situation awareness into conventional recommendation techniques (Chen, 2005; Karatzoglou et al., 2010). However, these approaches modify the algorithms themselves and thus cannot be applied to third-party recommendation libraries without modification of the libraries' source code. In our architecture, conventional recommendation techniques are encapsulated in the 2D-Recommender component, which can be perceived as a black box.

To consider contextual information in recommender systems without modifying the recommendation algorithms themselves, Adomavicius and Tuzhilin (Adomavicius and Tuzhilin, 2008) suggest the use of pre- and post-filtering. In our architecture, pre- and post-filtering is controlled by constraints and hints that are determined by the request refinement step. Constraints and hints represent characteristics of items that are suitable and relevant for the given set of situations. Furthermore, we added a particular situation selection step, which is required to handle the incoming live data in form of temporary situations.

The strengths of combining knowledge-based recommendations with CBF and CF have been exploited by researchers in the past, mainly with the goal to reduce the so-called cold start problems CBF and CF suffer from. Different hybridization methods exist to integrate knowledge-based recommendation techniques with CBF and CF (Burke, 2002). In a sense, our architecture can be perceived as an application of the *Cascade Method*: Pre-Filter, 2D-Recommender, and Post-Filter form a cascade in which the subsequent component refines the recommendations computed by the previous component.

6 CONCLUSION

In this paper, we have presented a software architecture for recommender systems that enhances arbitrary 2D recommendation techniques with situation awareness. We put forward a technology-independent recommendation process that yields recommendations adapted to both the user's preferences and current situation.

Our approach has several benefits. It provides context and situation awareness: domain knowledge and situational knowledge is used to prevent unsuitable recommendations (pre-filtering) and to foster recommendations that fit the current situation especially well (post-filtering). Furthermore, established and mature recommendation libraries can be used out of the box. Because our approach explicitly refrains from modifying recommendation techniques such as CBF and CF themselves, these libraries can be used without adapting their source code. This way, implementation costs can be reduced.

Moreover, we described a real-world case study that uses CEP and Semantic Web Technologies to implement the proposed architecture. The use of RDF, Jena Rules, and SPARQL results in a flexible application that can be adapted and extended easily by maintaining the domain ontology and refinement rules.

User feedback allows recommender systems to

understand a user's personal preferences. Storing the situations in which the user expressed his (dis-)satisfaction with a certain recommendation could allow the system to learn in which situations this specific recommendation is appropriate. Future research efforts should investigate how our approach can be extended to recommend items to the current user that other users in similar situations have liked. Extensions of our architecture should adhere to the current black box approach and not require a reimplementation or modification of existing recommendation algorithms.

REFERENCES

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749.

Adomavicius, G. and Tuzhilin, A. (2008). Context-aware recommender systems. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 335–336, New York, NY, USA. ACM.

Bouzeghoub, A., Do, K. N., and Wives, L. K. (2009). Situation-aware adaptive recommendation to assist mobile users in a campus environment. In *2009 International Conference on Advanced Information Networking and Applications*, pages 503–509.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

Chen, A. (2005). Context-aware collaborative filtering system: Predicting the user's preference in the ubiquitous computing environment. In *Proceedings of the First International Conference on Location- and Context-Awareness*, LoCA'05, pages 244–253, Berlin, Heidelberg. Springer-Verlag.

Ciaramella, A., Cimino, M. G. C. A., Lazzerini, B., and Marcelloni, F. (2009). Situation-aware mobile service recommendation with fuzzy logic and semantic web. In *2009 Ninth International Conference on Intelligent Systems Design and Applications*, pages 1037–1042.

Dötterl, J. (2016). Situation-aware recommender systems. Master thesis, Hannover University of Applied Sciences and Arts, Germany.

Hanani, U., Shapira, B., and Shoval, P. (2001). Information filtering: Overview of issues, research and systems. *User Modeling and User-Adapted Interaction*, 11(3):203–259.

Harris, S., Seaborne, A., and Prud'hommeaux, E. (2013). Sparql 1.1 query language. *W3C Recommendation*, 21.

Hermoso, R., Dunkel, J., and Krause, J. (2016). *Situation Awareness for Push-Based Recommendations in Mobile Devices*, pages 117–129. Springer International Publishing, Cham.

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S. (2009). Owl 2 web ontology language primer. *W3C Recommendation*.

Karatzoglou, A., Amatriain, X., Baltrunas, L., and Oliver, N. (2010). Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 79–86, New York, NY, USA. ACM.

Lops, P., de Gemmis, M., and Semeraro, G. (2011). *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. Springer US, Boston, MA.

Luckham, D. C. (2001). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Renners, L., Bruns, R., and Dunkel, J. (2012). Situation-aware energy control by combining simple sensors and complex event processing. In *Proceedings of the Workshop on AI Problems and Approaches for Intelligent Environments (AI@IE 2012 in conjunction with ECAI 2012), Montpellier, France, CEUR-WS.org*, volume 907, pages 29–34.

Rizou, S., Häussermann, K., Dürr, F., Cipriani, N., and Rothermel, K. (2010). A system for distributed context reasoning. In *2010 Sixth International Conference on Autonomic and Autonomous Systems*, pages 84–89.

Schelter, S. and Owen, S. (2012). Collaborative filtering with apache mahout. *Proc. of ACM RecSys Challenge*.

Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2.

World Wide Web Consortium (2014). Rdf 1.1 primer.

Ye, J., Dobson, S., and McKeever, S. (2012). Situation identification techniques in pervasive computing: A review. *Pervasive Mob. Comput.*, 8(1):36–66.