# New Challenges in Smart Campus Applications

Attila Adamkó, Tamás Kádek, Lajos Kollár, Márk Kósa, János Pánovics
Faculty of Informatics
University of Debrecen
Kassai út 26, H-4028 Debrecen, Hungary
Email: {adamko.attila,kadek.tamas,kollar.lajos,kosa.mark,panovics.janos}@inf.unideb.hu

***Abstract*—Nowadays very common keywords are Big Data, IoT (Internet of Things), crowdsourcing and ubiquitous computing. All of them gained greater emphasis and our University Campus is a great place where all of these areas could be investigated. Wide ranges of data could be collected from the built-in sensors of the building and naturally, from the users smartphones or tablets resulting a huge amount of data.**

**On one hand, our paper includes a framework which could provide value-added services for various people living or working on the Campus. On the other hand, the Campus is a perfect place where new algorithms could be developed and tested through these services. Furthermore, the community could be involved not just as subscribers for the services but also as providers of the data, and in an optimal case, the crowd could prepare and provide new information sources.**

***Index Terms*—campus, smart, adaptive, intelligent systems, crowdsourcing**

## I. Introduction

OUR primary goal was to create an architectural framework which allows various members of the community to create and use services based on the data that is collected in a university environment. These data include information on course enrollments, timetable, exam dates, office hours, and various deadlines along with community provided data. These collected data can then be subject to analysis, based on which they can either be fed back into the services, or we can provide recommendations or offer new services for our users.

Future Internet research—which was appeared at least seven years ago—aims at bridging the gap between both academic and industrial community's visionary research and large-scale experimentation [11], [2].

One part of it is the Internet of Things (IoT) phenomenon which highlights the opportunities lying in the sensors connected through wireless connections. We have successfully applied it one of our scenarios where the location of the users is crucial. Moreover, these sensor data serve as an endless source of environmental data which could drive a real-time and/or a transactional analytical module. In our vision this could be used to create trajectories and help users to find nearby colleagues or friends based on historical presence data and realtime sensor information.

The next piece of the trends is expressed by the term of ubiquitous computing (ubicomp) which states that computing could appear anywhere and everywhere. Borders are blurred between computing devices including desktop computers, notebooks, tablets and smartphones. Our framework currently includes a mobile and a desktop version which makes available the seamless and smooth usage of the system. Naturally, it is not mean the interface is the only thing which need to be available on the different devices. User profiles are created to support context-aware and customized environments for the ongoing research.

The third pillar of our framework is the crowd. Crowdsourcing could be applied in a University Campus as there are lots of people (students and staff) with different interests and different requirements for the services. However, they are not only consumers of the information produced by the system, they are producers—and content generators—as well. Lots of data are generated while various applications are used by them.

## II. Smart Communities

In this paper, when we explain the concept of the smart community we concentrate on the expectations connected with the software operating in a community. In our approach, the smart community is such a community that is served by smart applications. Based on the definition of the "Apps for Smart Cities Manifesto" for Smart Cities [7], the requirements of the smart community applications are the following:

- sensible—the environment is sensed by sensors;
- connectable—networking devices bring the sensing information to the web;
- accessible—the information is published on the web, and accessible to the users;
- ubiquitous—the users can get access to the information through the web, but more importantly in mobile any time and any place;
- sociable—a user can publish the information through his social network;
- sharable—not just the data, but the object itself must be accessible and addressable;
- visible/augmented—make the hidden information seen by retrofitting the physical environment.

An application itself, which satisfies partly or fully these requirements, could not be called smart at all. The basic requirement of a smart service to have information about the community. First of all, we have to collect and publish the available information. At the current level of technical development, the collected information is very huge and heterogeneous. A smartphone with average performance is also capable of GPS-based localization, light detecting, making photos, detecting of mobile and wireless network devices, etc.

Of course, the sensors are not always in the devices of the end users, think about of, e.g., a handle for free parking spaces for vehicles, a digital temperature sensor on buildings or an electronic passing gate system using RFID cards. These are all such kind of information which could be used to drive the creation of new services.

In many cases, the problem is that we have too much information. These information, temporarily, must be stored on the device that collects the data. The goal is, of course, the publishing of these data as soon as possible. The stage of the publishing is the web. In this place, we do not explain that it is practical to aggregate and to filter the collected data, moreover sometimes this procedure is mandatory due to the supporting of the anonymity [3]. Here let it be enough that the collected data must be maintained in a centralized, or, moreover, in a unified way, because of these data make the base of the smart services up.

The smart community is not static, it is continuously changing, and it is persistently on the move. The aspect and the amount of the collected information is changing from time to time. So, the evolution of the smart applications is a neverending process. New applications could be created, and their pure existence—the information about how we could use them—also could improve the amount of the sensible things. With this, we could collect new data about the working of the community. From the new data, we could know something about the behavior of the community, again, and these data could be reused in the life of the community.

## III. The extensible architecture

From the Smart Campus perspective, one of the main challenges is to collect content from various sources where the majority of them might be created at a later time. That is why we need a highly extensible system which is designed for change.

Such an extensible architecture has been designed and published by the authors in [1]. It allows the extension of the system with new elements on both the data producer and consumer side. This is where the crowd could help us by adding new sources and new services with the development of their own information parsers.

According to [1], a Smart Campus environment has lots of various (and most importantly, heterogeneous) data sources including the following:

- an Education Administration System called Neptun that contains information on course enrollments, timetable information of courses, exam dates and times, etc.,
- faculty members offering office hours, consultations, etc.,
- Education Offices of the various faculties offering office hours,
- Student Governments organizing events for students,
- the menus of the canteens located at the Campus,
- geolocation (e.g., GPS), WiFi or some other sensor data collected by smartphones or similar devices,
- data gathered by environmental and building sensors (temperature, humidity, air pressure, air pollution, etc.),
- a Library Information System that is able to tell whether a given book is available or not,
- social media sites (like Facebook or Google+) containing information on friends and ranges of interests of a person,
- professional sites (like LinkedIn) holding data on work experience and professional achievements (however, this is not necessarily the most important data source from Smart Campus perspective),
- bibliographic databases (like Google Scholar, DBLP or Scopus) that provide information of published journal articles or conference papers of researchers,
- event hosts of actually any events (like public lectures, concerts, exhibitions or whatever users might be interested in), and, which is essential,
- the crowd itself with the added value of the capability of generating content that is interesting for a set of people (or, to be more precise, consumers).

These are only examples of data sources not an exhaustive list. These demonstrate that what kind of diversity in data sources should a complex application face. Applications that provide value-added services typically require integration of some data coming from more of these data sources. That was the reason of developing an architecture providing the ability of accessing information from existing sources along with making the addition of new sources possible and (relatively) easy.

Some of the data can be collected in an automated way (e.g., sensor data), some others might require manual interaction (like canteens' menus or office hours of instructors); some of the data sources offer Application Programming Interfaces (APIs) to provide access to data (e.g., social media sites) while others do not have APIs therefore web crawlers are needed to gather and parse the data; some of the data sources provide built-in notification mechanisms (e.g., an event feed of a social network site) while others do not (for example, adding new office hours or changing the daily menu).

We have chosen the Extensible Messaging and Presence Protocol (XMPP) as the underlying communication protocol [9], due to its extensibility and publish/subscribe model. Further considerations on the design of the architecture are described in [1].

The power of our architecture as it does not limit the possible data sources and also allows the collection of some specific information. For example, if a couple of students prefer to have a lunch at the small restaurant near the Campus they can develop a connector that parses the restaurants web page to provide information on the daily menu. The same case can be also true for news feeds. When these sources are became available the potential (interested) users could be notified about it based on the preferences and the meta information provided for the feeds.

The Smart Campus Central Intelligence (SCCI) component in our architecture provides an interface between the information sources including both the incoming events (XMPP server) and the information stored in the database and the Web services layer (Figure 1).

It provides a couple of unified data models related to some of the most notable domains in a life of a Campus: educational data, research data, social data (with friend of and classmate of relationships), etc. SCCI layer also allows to define and

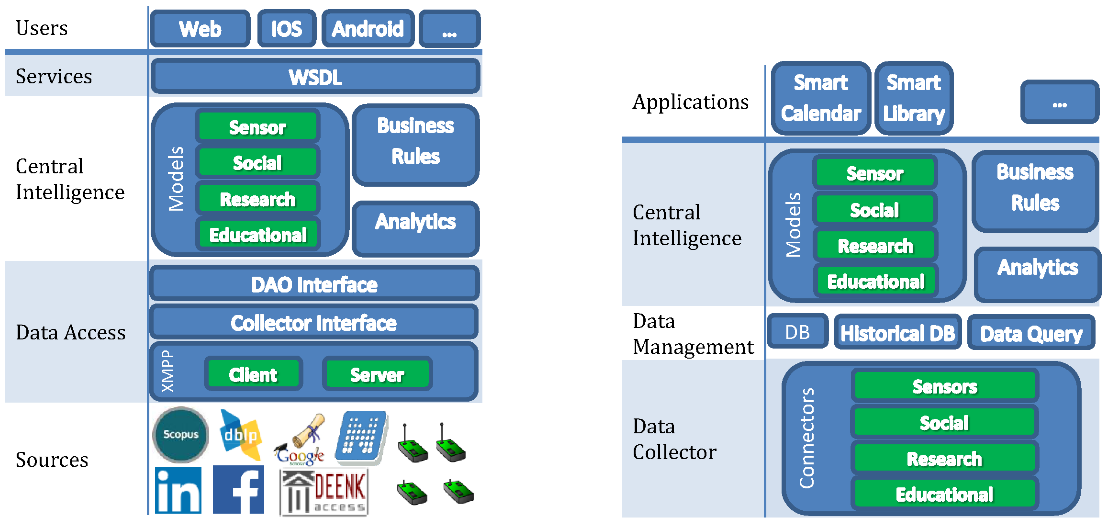


Fig. 1: Smart Campus Layers and Services

validate business rules and this is where the Analytics module resides. Now we have only a couple of simple analysis that does not exceed the scale of a medium-level database query but deeper analytical capabilities are planned to be added as this is how we can provide better value-added services.

An example where our services are created and used by the Smart Campus is the Faculty's portal which is based on Liferay. Its architecture is composed of three layers: Enterprise layer, Service layer and Persistence layer. While the Enterprise layer is responsible to fulfill all the enterprise needs (content, workflow, document, user, etc. management) the Service layer is the core component. It contains the majority of the business logic that perform enterprise needs. Liferay follows a Model Driven Architecture and this is made possible by the implementation of Service Builder which is used to automate the creation of interfaces and classes for database persistence and a service layer. In Liferay Web services has two mostly used and significant protocols: JSON Web Services and SOAP.

We have successfully implemented services in the portal to serve the lecturers' opening hours. One connector is used inside the Educational part to periodically check the available data. If there are changes the SCCI is notified. Based on the actual data all the affected users are notified about the change. Currently we are working on the extension of this portal service to alter it to a fully functional XMPP client. After that our portal service can directly notify the SCCI about the changes and the requested operations could be done without requiring the intermediate crawler.

An other ongoing development is the creation of a notification portlet that is integrated to the Faculty portal where users can see each other's online presence and start conversations without leaving the site. This is the advantage of the XMPP protocol and the usage of LDAP. It is straightforward because we can provide a platform where all the education-related tasks could be done therefore users can experience the added value of the services. Moreover, messages are not only sent between two users, but as an extension it is possible to have SCCI be a friend of all users. This gives the possibility of notifying users within the portal system, without the need of the execution of any third-party clients. It is similar to what we have as push notifications on the mobile platform. The portal changes its information provider role to an online collaboration environment.

### A. *The life cycle of intelligent services*

From our point of view, the basis of the intelligent community is to make the services online available. The first step is to collect the useful information into one online accessible database. First, it means a simple data service. For example, it could be a simple collection of news, a schedule, or even a timetable. In the second step, we have to pay attention on the use of available information. Notice the most popular news or columns, and then we can reorganize the information such as order the most popular columns to the first page. It is still not an intelligent service, but if we have a lots of application working together, which can share the information in the mentioned way, the service goes to be smarter and smarter.

## IV. Basic elements for adaption

Adaptive systems have gained increased popularity in the last decade. The overall goal is to improve the usability of the system and provide better user experience by applying personalization based on the services discussed in the previous section.

In a more technical view, we need to study and understood the structure of the metadata (i.e. the semantics) at the content's side which could fuel the adaptation process based on the user profile. Semantic markup included in the whole process with machine-understandable representation of the content. The work presented in this paper based on the previously mentioned extensible architecture where the Data Management layer contains the databases that are populated with data gathered by connectors of the underlying layer (see Figure 1). As of today, due to the characteristics of the data we have both a relational and a graph database as a backend. Well-structured information (e.g., course and timetable information) are stored in a normalized relational database. However, for semistructured or unstructured information the rigid structure of relational tables are not appropriate, so we have selected neo4j as a solution.

The result is a highly semi-structured system. It is not just collecting large volume and variation of date but also understanding the relationship between entities by mapping their connection into our system. With this method we established a data store which can be easily extended. When a new—and previously nonexistent—source is attached to the system the only task we need to perform is to add the new nodes and proper edges to the database.

In this context, following the Semantic Web initiative [10], ontology-based annotation helps us to provide adaptive processes, meaning that the incoming content enriched by semantic data. One could imagine all of these as tags attached to the data which could be a JSON message, an HTML fragment or a Web Service call. We need to transform this data into triples. The available prefixes are defined in the ontology into which these triples are going to be imported. Traditional SPARQL could be used to made queries, but we have found that graph databases also could be applied for reasoning.

Along with the semantic tags, the next important piece in our system are the groups. These groups modelling the subscribers which are ground for the original architectural idea that based on the publish/subscribe model. Groups are high level entities and some of them are automatically created, like the group for a user's personal calendar or for the news feeds, etc. As a member of a given group you will receive the events published by that source, and naturally one user can join as many he/she wants and one may create new ones too. The visibility of the groups also could be controlled, there could exist public and private groups where freely everybody or only the invited members could join.

## V. Smart Campus applications

### A. Adaptive Event Recommendation as a Personal Calendar

The first application which appeared in the concept of the Smart Campus at University of Debrecen was a simple data serving application gathering into one location all the important events at the Faculty. The first principle—following the outlines of the Intelligent application's lifecycle—was only to provide a uniform Web Service interface to made accessible all of that data. It could open the way to all of the end-user services, like reminding for important deadlines or browsing the categorized events.

However, collecting the information from distinct information systems and providing them through a standardized way is a useful approach—but cannot be seen as an intelligent one. On the contrary, the usage scenarios of the published information could serve as a base for predicting their behavior. Take the example, users could mark events as important to create their personal calendar. The system could analyze these marks and highlight the most frequent and important events. While checking the marks the system could detect that some of the events are closing to the physical limit of the room where it is scheduled. In that case, searching for a greater room is essential and required. Finally, the system could notify all the attendees from the change of the location. Naturally, the notification is based on the previously mentioned group-based solution.

In that calendar we have developed the possibility to browse, categorize and register for events. Hereafter, we made available to rate those events. The service is available on the following address: http://smartcampus.hu.

An Android application (see Figure 2) has also been implemented to support easier access of those services. This mobile application allows additional services to use. As we have stated earlier, the base of an intelligent service is not just the direct information posted by the users but may origin from sensors as well. We investigated the possibilities in our building and prepared an application for smartphones which could determine with a very good approximation the position of the user based on the WiFi network access points. When connecting this data with the user's personal calendar, we could show the nearby events in the first place on the list. The demonstration of this service is planned for a local conference held in this autumn where the application will be used to show the nearby sections program at first place.

Moreover, the event recommendation system is not based just time and place attributes. The categorization made it available to recommend events for the user based on its topics. Imagine the situation when you mark important events and the system recommends you upcoming possibilities based on the categories—and (ontology based) related categories—of the marked events.

### B. Adaptive Meeting Planning

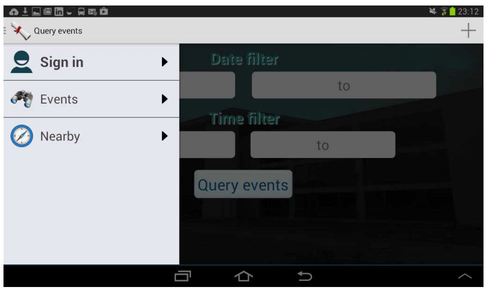
For the above mentioned calendar we have an ongoing development to extend it with a meeting planning module. That module will read all the participants calendar events and try to suggest time slots which could be appropriate for all the attendees.

The adaption is based on properties of the event which is related to the user as well. It could be mandatory, optional or rescheduleable—like a registration for an opening hour but the meeting may be scheduled to the same time because one can attend an other opening hour at a later time [6].

### C. Managed Programming Contests

The basic goal of the ProgCont system was to support the organization part of programming contests. The development process started in 2011 with a web application and worker services. The web application stores the exercises in a problem catalogue and collects the submissions including not only the solution source code, but the necessary information about the competitors and contests. The worker services are used to evaluate the submissions, they compiles them (if it is possible), and validate the compiled algorithm by running several test cases.

Of course the software itself could not be called smart at all, even if most of the exercises were selected from international programming contest for the purpose of exercising our students. But the ProgCont system collects numerous additional



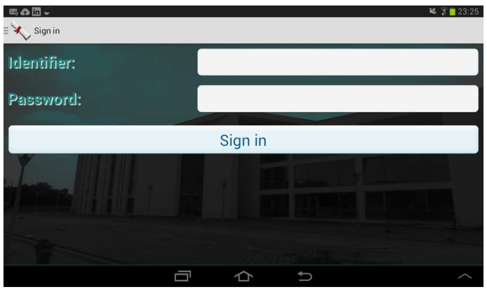


Fig. 2: Smart Campus Android application

data during its operation, which give us the ability to discover extra knowledge for the users [8].

One trivial example based on the success rate of submissions, more precisely the number of users who can solve the exercise, and the number of trials before the first successful solution. Everyone knows own success rate. The additional information in that case means that we made the cumulated results available.

Another useful information could be the quality measure of the accepted solutions. It means, that we measure and rank the accepted solutions using another criteria such as the length of source code, the required amount of memory, the execution time, the number of iterations, and so on.

There exists several another way to support our student with the ProgCont system. A catalog of thematic exercises are also available, giving the possibility to browse problems with the same algorithmic background. ProgCont can provide automatic suggestions if the necessary amount of information is available about the user, in other words, when there is enough evaluated submission to approximate the user skill.

One objective way to measure the worth of an exercise is to calculate the rate of accepted and unsuccessful submissions. But in the other hand, the users can also rate the exercises. Both ways can serve useful information for further decisions about how should the user continue the preparation process.

Nowadays the system also used in classroom test. The collected data gives the ability to do statistical investigation from the pedagogy point of view. Relative frequency histogram and relative frequency of scores are available according to the tested groups, and the connection between separate classroom tests can be discovered using a diagram representing correlation.

### D. *Assessment system for non-graded exercises*

A Spring-based Web application with a JavaServer Faces (JSF) frontend has been developed in order to help course instructors in offering optional exercises or assignments (which do not count into the grade) for their students. In principle, these exercises can be of any kind (programming, database, math, etc.) but we primarily focused on programming exercises. The primary difference from ProgCont is that unlike ProgCont this system does not require any feedback from the instructor side. It was designed in order to decrease the load on instructors by asking the crowd (i.e., other students) to validate submissions (that is why it is only used for optional exercises). Unfortunately, there are some courses at our institute where we have relatively few course instructors. Their number is enough for creating and validating assignments that count into the grade but short for doing the same for not graded exercises (e.g., they did not have enough time for preparing appropriate test cases in order to validate with ProgCont).

The Web application offers the following major functionalities. It is possible

- to define exercises,
- to submit solutions,
- to assess submissions,
- to comment and/or rate assessments.

Any registered user (i.e., even students) can define exercises. Those who solve an exercise can submit the solution which can be evaluated (assessed) by anyone else. Therefore practising students can receive some feedback, even if these answers cannot be trusted. To deal with false positive feedbacks, it is important that not only the solutions but the people giving feedbacks should also be rated. Later, those people's ratings who regularly give wrong answers will count less.

Similarly to ProgCont, the data gathered by this application can (and should) also be analysed using data mining techniques. Currently, assessments are created on a voluntary basis. However, good assessments will come from qualified people who understand several aspects of the submitted solution. It is very hard to find suitable reviewers whose knowledge level and experience is sufficient for providing valuable assessments. After analysing the gathered data we can classify possible reviewers based on how valuable their assessments are therefore the system will be able to offer a set of possible reviewers for each task.

## VI. A SAMPLE CALENDAR SERVICE PROBLEM

Some of the problems that arise concerning Smart Campus applications might be solved using artificial intelligence methods. In this chapter, we present a sample problem related to the calendar service. In order to define the problem, we first introduce the Extended State-Space Model, then a state-space representation of the problem is given. Later, we summarize the Extended Breadth-First Algorithm (EBFS).

### A. *The Extended State-Space Model (ESSM)*

The EBFS algorithm can be defined after introducing an extended state-space model [4], [5], which allows us to discover the representation graph starting from several different states and possibly in more than one direction. Using state-space representation, solutions to problems are obtained by executing a series of well-defined steps. During the execution of each step, newer and newer states are created, which form the state space. States are distinguished from one another based on their relevant properties. Relevant properties are defined by the sets of their possible values, so a state can be represented as an element of the Cartesian product of these sets. Let us denote this Cartesian product by $S$. Possible steps are then operations on the elements of $S$. Let us denote the set of operations by $F$. The state space is often illustrated as a graph, in which nodes represent states, and edges represent operations. This way, searching for a solution to a problem can be done actually using a path-finding algorithm.

We keep the basic idea (i.e., the concepts of states and operations on states) also in the extended state-space model (ESSM). The goal of this generalization is to provide the ability to model as many systems not conforming to the classical interpretation as possible in a uniform manner.

A state-space representation over state space $S$ is defined as a 5-tuple of the form

$$\langle K, \text{initial}, \text{goal}, F, B\rangle,$$

where

- $K$ is a set of initially known (IK) states, such that $K \subseteq S$ and $K \neq \emptyset$,
- $\text{initial} \in \{\text{true}, \text{false}\}^S$ is a Boolean function that selects the initial states,
- $\text{goal} \in \{\text{true}, \text{false}\}^S$ is a Boolean function that selects the goal states,
- $F = \{f_1, f_2, \ldots, f_n\}$ is a set of "forward" functions, $f_i \in (2^S)^S$,
- $B = \{b_1, b_2, \ldots, b_m\}$ is a set of "backward" functions, $b_i \in (2^S)^S$.

The "forward" and "backward" functions represent the direct connections between states. For more details, see [4].

Some notes:

- The number of initial and goal states is not necessarily known initially, as we may not be able to or may not intend to generate the whole set $S$ before or during the search.
- The $n + m = 0$ case is excluded because in that case, nothing would represent the relationship between the states.
- Although the elements of the sets $F$ and $B$ are formally similar functions, their semantics are quite different. The real set-valued functions in $F$ are used to represent nondeterministic operators, while there may be real set-valued functions in set $B$ even in case of deterministic operators.

Let us now introduce a couple of concepts:

- *Initial state*: a state $s$ for which $s \in S$ and $\text{initial}(s) =$ true.
- *Goal state*: a state $s$ for which $s \in S$ and $\text{goal}(s) = \text{true}$.
- *Known initial state*: an initial state in $K$.
- *Known goal state*: a goal state in $K$.
- *Edge*: an $\langle s, s', o\rangle \in S \times S \times (F \cup B)$ triple where if $o \in F$, then $s' \in o(s)$, and if $o \in B$, then $s \in o(s')$.
- *Path*: an ordered sequence of edges in the form

$$\langle s_1, s_2, o_1\rangle, \langle s_2, s_3, o_2\rangle, \ldots, \langle s_{k-1}, s_k, o_{k-1}\rangle,$$

  where $k \geq 2$.

General objective: determine a path from $s_0$ to $s^*$, where $s_0$ is an initial state, and $s^*$ is a goal state.

## B. A Problem

One of the first applications developed in Smart Campus is a special service processing calendars that contain some events marked by students as important. If some of these events conflict in time with each other, the calendar service may suggest another schedule by replacing the time intervals of some events with other possible intervals. Suppose the students designates a schedule containing $k$ events ($E_1, \ldots, E_k$), each with an initial time interval (from $T_{1,1}, \ldots, T_{1,N_1}$ to $T_{k,1}, \ldots, T_{k,N_k}$). The problem, which can be solved with the use of EBFS, is to determine a schedule of the same events such that no two time intervals are in conflict.

*1) State Space:* In this state-space representation, a state of the problem is represented by a $k$-tuple, the elements of which describe the currently set time intervals of each event. The IK states are arbitrarily chosen by the user and may contain interference between the time intervals of the events. Our goal is to eliminate this interference. In this model, initial states have no significance.

$$E_1 = \{T_{1,1}, \ldots, T_{1,N_1}\}, \ldots, E_k = \{T_{k,1}, \ldots, T_{k,N_k}\}$$

$$S = \{\langle t_1, \ldots, t_k\rangle \in E_1 \times \ldots \times E_k \ : \ t_j \in E_j\}$$
$$K = \{\langle t_1, \ldots, t_k\rangle\} \subseteq E_1 \times \ldots \times E_k$$
$$\text{initial}(\langle t_1, \ldots, t_k\rangle) = true$$
$$\text{goal}(\langle t_1, \ldots, t_k\rangle) = \begin{cases} true & \text{if } \forall p \forall q\, (p \neq q \rightarrow t_p \cap t_q = \emptyset) \\ false & \text{otherwise} \end{cases}$$

*2) Operators Over the State Space:*

$$F = \left\{\text{update}(l, r) \in \left(2^S\right)^S\right\},$$
$$l \in \{1, \ldots, k\}, \quad r \in \{1, \ldots, N_l\}$$
$$\text{update}_{l,r}(\langle t_1, \ldots, t_k\rangle) = \langle t'_1, \ldots, t'_k\rangle$$
$$\text{where} \ \ t'_j = \begin{cases} T_{l,r} & \text{if } j = l \\ t_j & \text{otherwise} \end{cases}$$
$$B = F$$

## C. The EBFS Algorithm

The EBFS algorithm extends the BFS algorithm with the ability to run more than one breadth-first search starting from more than one state (the initially known states).

The EBFS algorithm stores a subgraph of the representation graph during the search. The main difference from BFS at this point is that in case of EBFS, the relationship between the nodes and each IK state is stored.

The full pseudocode of the EBFS algorithm can be found in [4]. The database of the algorithm stores for each node the state represented by the node as usual, the forward and backward status (open, closed, or not relevant), forward and backward parents, forward and backward children of the node, as well as the distance from and to each of the IK states.

## VII. Conclusion and future work

In this paper we in traduced the intelligent services and outlined that the development of smart applications is a neverending process. The underlying service architecture are now in a testing phase and several end-user application prepared but several more need to be created before we could call our system smart. The architecture fits well into the more general publish/subscribe based architecture of Smart City and Smart Campus applications as its extensible with new data sources providing the capability of integration of heterogeneous data. In the future, development of the Analytics module is a major goal since providing good analysis of the collected data can add more value to the services.

## Acknowledgements

The publication was supported by the TÁMOP–4.2.2.C–11/1/KONV–2012–0001 project. The project has been supported by the European Union, co–financed by the European Social Fund.

## References

[1] Attila Adamkó and Lajos Kollár. Extensible data management architecture for smart campus applications—a crowdsourcing based solution. In *WEBIST (1)*, pages 226–232, 2014.

[2] Y. Atif and S. Mathew. A social web of things approach to a smart campus model. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 349–354, Aug 2013.

[3] Mikel Emaldi, Oscar Pena, Jon Lazaro, Diego Lopez-de ipina, Sacha Vanhecke, and Erik Mannens. To trust, or not to trust: highlighting the need for data provenance in mobile apps for smart cities. In *International Workshop on Semantic Sensor Networks, Proceedings*, pages 1–4, 2013.

[4] Tamás Kádek and János Pánovics. Extended breadth-first search algorithm. *International Journal of Computer Science Issues*, 10(6):78–82, 2014.

[5] Tamás Kádek and János Pánovics. Some improvements of the extended breadth-first search algorithm. *Studia Universitatis Babeş-Bolyai, Informatica*, 59(Special Issue 1):165–173, 2014.

[6] Haim Kaplan, Ilia Lotosh, Tova Milo, and Slava Novgorodov. Answering planning queries with the crowd. *Proc. VLDB Endow.*, 6(9):697–708, July 2013.

[7] Ingo Lütkebohle. The Apps for Smart Cities Manifesto. http://www.appsforsmartcities.com/?q=manifesto, 2012. [Online; accessed 15-December-2014].

[8] Adam Marcus, David Karger, Samuel Madden, Robert Miller, and Sewoong Oh. Counting with the crowd. *Proc. VLDB Endow.*, 6(2):109–120, December 2012.

[9] P. Saint-Andre. RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core., March 2011.

[10] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[11] Róbert Szabó, Károly Farkas, Márton Ispány, András A. Benczúr, Norbert Bátfai, Péter Jeszenszky, Sándor Laki, Anikó Vágner, Lajos Kollár, Csaba Sidló, Renató Besenczi, Máté Smajda, Gergely Kövér, Tamás Szincsák, Tamás Kádek, Márk Kósa, Attila Adamkó, Imre Lendák, Bernát Wiandt, Timon Tomás, Ádám Nagy, and Gábor Fehér. Framework for smart city applications based on participatory sensing. In *Proceedings of the $4^{th}$ IEEE International Conference on Cognitive Infocommunications*, pages 295–300, Dec 2013.