Intelligent Calendar Applications: A Holistic Framework based on Ontologies

K. Agnantis¹, A. Alexiadis¹ and I. Refanidis¹

¹Department of Applied Informatics, University of Macedonia, 54006, Thessaloniki, Greece; Email: kagnadis@gmail.com, talex@java.uom.gr, yrefanid@uom.gr

ABSTRACT

With the term ‘Intelligent Calendar Application’ we refer to traditional electronic calendars empowered with scheduling engines and rich information about events, in order to be able to automatically schedule the events in time and space. The additional information about events that is necessary to automate their scheduling includes temporal domain, duration range, feasible locations, constraints and preferences over the way the events are scheduled, etc. Asking the user to provide all this information for each event would render intelligent calendars less attractive compared to traditional calendars. In order to overcome this limitation, we have recently proposed the use of general ontologies to describe everyday human activities. In this article we extend our work in three ways: (a) by extending the general ontologies with domain ontologies, particularly, an ontology to describe activities related with actively participating in scientific conferences; (b) by integrating these ontologies within a deployed intelligent calendar application; and (c) by developing a module to automatically extract ontological activity descriptions from web pages of scientific conferences. In this way, we propose a holistic framework concerning the integration of ontologies within intelligent calendar applications, comprising information modelling, gathering and integration.

Keywords: Ontologies; electronic calendars; intelligent systems; information extraction.

Mathematics Subject Classification: 68T30, 68T35, 68U35

Computing Classification System: I.2.4, H.4.1

1. INTRODUCTION

Traditional electronic calendars, like Google Calendar, Yahoo! Calendar or Microsoft Outlook use the iCalendar¹ standard to describe events, where an event is considered an entry within the user’s calendar. The iCalendar standard assumes that an event is already scheduled, so the attributes supported to describe events include its start time, its duration, a single text-based location, a title and a description, the list of participants (if any), etc. The applications are characterized by rich user interfaces to facilitate their use, whereas they provide additional functionalities, such as calendar sharing, automated notifications, etc. However, neither the iCalendar standard, nor the traditional electronic calendars have any provision for automatically scheduling events.

With the term ‘activity’ we commonly refer to traditional events, that is, scheduled commitments like a lecture, as well as to tasks, that is, unscheduled commitments with a deadline such as preparing the slides for a presentation. So, we expect from the next generation electronic calendar applications,

¹ https://tools.ietf.org/html/rfc5545
commonly referred as intelligent calendar applications (ICA), to operate like decision support systems, that is, helping the user to schedule his daily activities effectively and efficiently. This can be achieved by employing scheduling technology and solving the underlying computational problem on behalf of the user. Rescheduling should also be supported, in case new activities arise or existing activities change.

In order to automate user activities scheduling, a rich model about them is necessary. For each human activity the scheduler needs to know a plethora of details, such as when it can be scheduled, which are the feasible locations where the user should be in order to perform it, what are its minimum and maximum duration, whether the activity can be executed in parts or at once, whether there are constraints between this activity and others, potential user preferences about how the activity is scheduled in time and space, as well as in relation to each other, and many others.

Asking the user to manually provide all this information would be very cumbersome and would discourage him from adopting an ICA to organize his time, even if sophisticated user interfaces are employed to gather this information [Alexiadis and Refanidis, 2009]. So, here is where ontologies can contribute significantly [Horrocks, 2013]. Ontologies can be adopted by ICAs to describe human activities, including all the aforementioned details that are necessary to model the scheduling problem rigorously. Furthermore, using domain dependent ontologies, activity descriptions ready to insert in ICAs could be provided, in the same way nowadays such descriptions are provided in iCalendar format by several event organizers. So, it would be possible for the user of an ICA to add new activity descriptions in his activity list with just a few clicks.

Recently, we have proposed the use of three general ontologies to describe the scheduling aspects of human daily activities [Agnantis and Refanidis, 2015b]. In this article, we elaborate over our previous work in three ways: (a) We extend the proposed general ontologies with a domain ontology, targeting activities related with participating in scientific conferences; (b) we extended a deployed ICA, particularly SELFPLANNER [Refanidis and Alexiadis, 2011], in order to adopt activity descriptions using these ontologies; and (c) we have developed a module to automatically extract ontological activity descriptions from scientific conferences web sites, that are ready to be inserted in any ICA supporting the proposed ontologies. In this way, we propose a holistic framework concerning the integration of ontologies within intelligent calendar applications, covering information modelling, gathering and integration.

As a motivating example for this article, consider the following: Kostas is a computer scientist and intends to submit a research paper to his favorite conference, as well as to present it himself, provided that it will be accepted. He starts by feeding his intelligent agent with the URL of the conference site. The agent extracts all the important dates and creates four activities concerning Kostas' conference participation. The first two activities concerns preparing and submitting a research paper. For this activity, the agent asks Kostas about his estimated duration to prepare the paper. The second activity concerns revising the paper and submitting the camera ready version of it. The third activity concerns preparing the paper presentation. And the fourth activity concerns attending the conference. For the last three activities, default values are used wherever needed (e.g., duration, preferences, etc.). Of
course, in case the submitted paper is not accepted, the last three activities have to be deleted from
Kostas’ calendar.

The rest of this article is structured as follows: Section 2 gives an overview of the related work
concerning activity ontologies and intelligent calendars. Section 3 briefly presents the latest version of
the proposed general activity ontologies, as well as the new domain ontology for scientific
conferences. Section 4 presents a new library, called ONTPLANNER, developed to allow the integration
of the proposed ontologies within the SELFPLANNER ICA. Section 5 presents the information extraction
module that extracts ontological activity descriptions from scientific conference sites. And, finally,
Section 6 concludes the article and identifies directions for future research in this area.

2. RELATED WORK

This section presents related work in two directions: activity ontologies and earlier attempts towards
intelligent calendar management.

2.1. Activity Ontologies

There are several relatively recent efforts to organize human activities using ontologies, although not
with the goal to insert them in an ICA. [Raimond and Abdallah, 2007] present an event model, with
each event being defined by attributes such as location, time, participants/agents, properties or
outputs. The Simple Event Model proposed in [Van Hage et al., 2011] tries to provide a simple RDF
model for describing any event based on its type (what), its location (where), its participants (who) and
its temporal information (when), allowing for a large category of questions to be answered using
simple SPARQL queries.

[Riboni and Bettini, 2011] use the Activity class as the root of their ontology and define a hierarchy of
different sub-activities based on specific rules that an activity applies to, e.g., the number of
participants (SocialActivity), or various characteristics of them (e.g CarnivalParty). Their work aims at
identifying indoor complex human activities, thus the ontology focuses on modeling activities for smart
home and sensor information. Another similar approach was followed in the EU Project Dem@Care
[Meditskos et al., 2014], where the activity ontology tries to represent simple every day human
activities or conditions (e.g., sleeping or watchingTV) and by composing them to produce new more
interesting and complex ones.

[Hu and Janowicz, 2012] propose a quite interesting and promising idea of defining various activity
ontologies that will be able to interconnect human activities in the real world (like writing a paper) with
computer events in the digital world (e.g., calendar events).

COURSER [Agnantis and Refanidis, 2015a], an intelligent system to schedule composite educational
objects, provides an ontology to describe educational activities, such as lectures, labs, homework,
midterm and final exams. The underlying ontology describes temporal aspects of the educational
activities through natural language and a domain specific grammar, with a parser being employed to
convert these descriptions in a more structured form. Similar is the situation with the MYVISITPLANNER
system [Refanidis et al., 2013], with an ontology being developed to identify various classes of atomic
cultural activities (e.g., museum visit).
2.2. Automating individual and joint activity scheduling

SELFPLANNER [Refanidis and Alexiadis, 2011; Refanidis and Yorke-Smith, 2010] is a recent intelligent calendar application focusing on individual activity scheduling (no automated meeting scheduling is supported). Among others, it supports:

- Variable activity duration, with more duration resulting in more utility.
- Arbitrary temporal domains and arbitrary utility functions over them.
- Alternative locations for each activity with geolocation information attached to them.
- Interruptible and non-interruptible activities, with additional constraints and preferences supported over the way the different parts of an interruptible activity are scheduled in relation to each other.
- Periodic activities, with each occurrence of them being scheduled separately within its period.
- Ordering, proximity and implication binary constraints and preferences over pairs of activities.
- Temporal overlapping between low-intense activities.

The scheduler solves a constraint optimization problem in two phases: In the first phase, it creates an initial seed plan, by using an adaptation of the well-known Squeaky Wheel Optimization [Joslin and Clements, 1999] framework. In the second phase, it tries to improve this plan, by employing stochastic local search [Alexiadis and Refanidis, 2013a; 2015]. Furthermore, the scheduler is able to produce alternative plans of high utility and significant deviation between each other [Alexiadis and Refanidis, 2013b].

Besides scheduling individual activities, there are several research efforts over the last two decades that focus on joint activities, that is, meetings. Older efforts attacking specific aspects of this problem include [Garrido and Sycara 1995; Jennings and Jackson 1995; Sen and Durfee 1994; Sen and Durfee 1998]. More recent efforts tend to incorporate learning components or to integrate with the Semantic Web. For example, RCal is an intelligent meeting scheduling agent that assists humans in office environments to arrange meetings [Payne, Singh and Sycara 2002; Singh, 2003]. CMRadar [Modi et al. 2004] is an end-to-end agent for automated calendar management that automates meeting scheduling by providing a spectrum of capabilities ranging from natural language processing of incoming scheduling-related e-mails, to negotiate with other users or making autonomous scheduling decisions.

PTIME [Berry et al. 2006] aims at facilitating meeting scheduling. The innovation of PTIME lies at its capability to learn the user's preferences thus adapting its future behavior, whereas it emphasizes in adopting natural language for interfacing with the user. Another effort concerns Towel [Conley and Carpenter 2007], an initial attempt towards an intelligent to-do list. Towel allows the user to organize to-dos (group, tag, check, etc) as well as delegate them to other users or agents.

3. ONTOLOGIES FOR INTELLIGENT CALENDARS

In this section we overview three general ontologies to describe activities with all the necessary details that facilitate their automated scheduling by ICAs. A more detailed presentation of these activities is provided in [Agnantis and Refanidis, 2015b].
Furthermore, we present a domain ontology, specifically designed for describing activities related with the various phases of a scientific conference. All ontologies exploit features from other established ontologies whenever this is possible. Particularly, they exploit the FOAF ontology [Brickley and Miller, 2012] to represent user related information, like names, addresses, contact info, etc; the OWL-Time ontology [Hobbs and Feng, 2004] to represent time related information, like duration, dates, intervals, etc; and the Geo OWL ontology\(^2\) to represent spatial information, such as the latitude or the longitude of a specific location on earth\(^3\).

### 3.1. The Activities ontology

The Activities ontology aims at describing any type of human daily activity (Figure 1). The most important class is the Activity class, which aims at representing semantically different types of activities performed in everyday life by ordinary people. For example, indoor or outdoor activities, joint or individual activities, activities with preconditions, etc. The ontology tries to distinguish all these different types and prerequisites, providing an easy and straightforward way to represent this kind of information.

Activity class has two direct subclasses: ComplexActivity and SimpleActivity. The ComplexActivity class is used to model groups of alternative (ComplexOneOfActivity) or required (ComplexAllActivity) activities. For example, Clean_House activity may be composed of several other sub-activities, like Wash_Dishes, Vacuum_the_floor and Do_the_laundry, rendering it a member of the ComplexAllActivity class. On the contrary, the Eat_Launch activity is a member of the ComplexOneOfActivity class, because it can be represented by either Cook_meal or Order_pizza, as the completion of the complex activity requires the completion of just one of the alternatives.

Class SimpleActivity, having as its single subclass HumanActivity, represents atomic activities that are performed by humans. Direct subclasses of HumanActivity are Work, Recreation, Leisure, Indoor, Outdoor, etc.

Furthermore, we have defined several dozens of objects that represent instances of some of the most common types of activities. Some examples are the following:

- Going_to_shoes_store (ShoppingActivity)
- Church_visiting (ReligiousActivity)
- Running (RecreationActivity)
- Attending_a_wedding (SocialActivity)
- Taking_exams (SchoolActivity)
- Painting (ArtActivity)
- Watching_tv (AudioVisualActivity)
- Card_game (GameActivity)
- Ironing (HousekeepingActivity)
- Playing_basketball (RecreationActivity)
- Sunbathing (BeachActivity)
- Visiting_theater (CulturalActivity)
- Underwater_photography (WaterActivity)

\(^2\) http://www.w3.org/2005/Incubator/geo/XGR-geo/#owl

\(^3\) All ontologies presented in this article are available at https://github.com/agnantis/uom-ontologies
Other classes of the ontology include the following:

- **Venue:** It represents the type of the location where the activity takes place. Its subclasses include general descriptions like PublicBuilding, Outdoor and EnterpriseHall, as well as more specific cases like UniversityCampus, BeachBar and Office.
- **GeoLocation:** This class defines the geo-coordinates of a location.
- **Participant:** This class represents the type of each participant in an activity. It could be one of the following values: \{Group, Person, Organization\}.
- **TimeInterval / DateInterval / DateTimeInterval:** These classes describe a time, date or date-time temporal interval respectively.

![Class hierarchy](image)

**Figure 1.** The top classes of the Activities ontology within Protégé

Auxiliary properties to describe an activity are the following:

- **hasVenue:** It is used to attach a venue to an activity.
- **hasScheduledTime:** It is used to attach temporal intervals to an activity.
- **hasDuration:** This property defines the duration of the activity.
- **hasTemporalWindow:** This property defines the temporal domain of the activity, in the form of a list of temporal intervals.
- **requires:** This multivalue property defines preconditions to accomplish an activity.
It is important to note that the punning technique [Vrandecic and Krotzsch, 2006] has been used, particularly with classes Activity and Venue, in order to define classes and objects with the same name. For example, JoggingActivity as a class comprises all the cases someone went for jogging; on the other hand, JoggingActivity as an object can be used to express that someone likes going jogging.

3.2. The Meetings ontology

The Meetings ontology aims at representing semantically all the information needed by two or more people to arrange a meeting, thus reducing the need for complex interactions towards reaching an agreement. The basic class is Meeting, with two subclasses, ArrangedMeeting and ToBeArrangedMeeting (Figure 2).

Each meeting may have one or more topics (property topic), as well as one or more alternative locations (property hasVenue). Object NoLocation denotes that the meeting is not bound to a specific location, e.g., in case of making a mobile phone call. Property hasParticipant denotes the tentative participants of the meeting.

![Figure 2. The top classes of the Meetings ontology within Protégé](image-url)
Arranged meetings are described through the properties hasStartTime, hasEndTime and hasDuration. On the other hand, not yet arranged meetings have the multivalue property hasTemporalWindow, allowing to define temporal intervals when the meeting can be scheduled. They also are described by a minimum (property hasMinDuration), a maximum (property hasMaxDuration) and a preferred (property hasPreferredDuration) duration.

The Participant class aims at representing the participants of a meeting. For each participant, we need to know his role (multivalue property hasRole), importance (hasPresenceType) and availability (property hasTimeProgram).

We do not dig further into the details of this ontology, since it is not further referenced within this article.

### 3.3 The CalendarPrefs ontology

The CalendarPrefs ontology aims at representing details about the user of an intelligent calendar, as well as his preferences over the way activities are scheduled (Figure 3).

![Figure 3. The top classes of the CalendarPrefs ontology within Protégé](image)

We present its classes in a bottom-up direction, starting from the auxiliary ones.

- **Class** Topic is used to represent the interests of a user, such as Technology, Health or Athletics.
- **The enumerated class** EducationLevel represents the educational level of the user. Its values (named individuals) are Primary, Secondary, College, University, etc.
• The enumerated class `ActivityExecutionType` consists of two member values, `NotPreemptiveExecution`, indicating activities that can only be executed at once, and `PreemptiveExecution`, representing activities that can be executed in parts.

• `Class ActivityType` represents activity types. It serves as a connection point between this ontology and the `Activities` one.

• The enumerated class `Gender` has two values, `Male` and `Female`.

• `Class LocationType` represents either specific locations (e.g., a user's home) or location types (e.g., football court). A member of the `LocationType` class can also be a member of `Venue` class.

• `Class Preference` is a general class that represents all sorts of preference. `ActivityPreference` is a subclass of it.

• `Class Profile` is a general class that represents all sorts of profiles. `Class UserProfile` is a subclass of it.

The `UserProfile` class represents the personal details of the user. This is achieved through a set of FOAF properties, such as `hasFirstName`, `hasLastName`, `hasNickName`, `hasGender`, etc, ranging over simple data types (strings and numbers) or over the auxiliary aforementioned classes.

The `ActivityPreference` class has several subclasses, depending on the type of preference over activities. The two subclasses, `UnaryPreference` and `BinaryPreference`, define preferences that concern single activities and pairs of activities respectively. The three subclasses `ModePreference`, `LocationPreference` and `TemporalPreference` define preferences that concern the mode in which the activity is executed, the location it is scheduled and the temporal way it is scheduled respectively.

A subclass of `ModePreference` is `ActionPreference`, defining whether an activity is preferred to be executed in preemptive or non-preemptive mode.

`LocationPreference` has two subclasses, `PreferLocation` and `AvoidLocation`, defining types of locations that the user prefers or does not prefer an activity to be scheduled there.

`TemporalPreference` has several subclasses, defining how an activity is scheduled in time. Particularly:

• `MaxFrequency` / `MinFrequency` / `PreferredFrequency`: These classes are used to define the number of repetitions of an activity over a temporal interval. For example, the user might want to go at most 3 times a week to the gym.

• `MaxTimeDistance` / `MinTimeDistance` / `PreferredTimeDistance`: These classes define the temporal distance between two activities.

• `MaxAccumulatedDuration` / `MinAccumulatedDuration` / `PreferredAccumulatedDuration`: These classes define the accumulated duration of an activity.

• `MaxPartDuration` / `MinPartDuration` / `PreferredPartDuration`: These classes are used to define the maximum / minimum / preferred duration of each part of a preemptive activity.
### 3.4 The *Conferences* ontology

The *Conferences* ontology aims at semantically representing all the important information of a scientific conference, like the starting and ending date of the main event, the various submission and notification deadlines, etc (Figure 4). The main class of the ontology is the *Conference* class. An individual of this class has the following properties:

- **name**: a string property holding the name of the conference.
- **location**: a string property holding the name of the venue.
- **startDate/endDate**: date properties holding the dates that the conference start and ends, respectively.
- **hasImportantDate**: an object property, holding *ImportantDate* individuals.

![Class hierarchy](image)

**Figure 4.** The top classes of the *Conferences* ontology within Protégé

An individual of the *ImportantDate* class is used to encode the information related to a specific important date of the conference. We have added several subclasses of the *ImportantDate* class, such as *Abstract*, *Paper*, *CameraReady*, *EarlyRegistration*, etc. There are two properties defined for the *ImportantDate* class (and all its subclasses):

- **submissionDate**: a date property for a submission date
- **notificationDate**: a date property for a notification date

Every *ImportantDate* individual should have exactly one submission date, while it can have one or no notification dates.

As an example, the following code represents the definition of a conference and some of its important dates (particularly abstract, paper and workshop deadlines):
ONTPLANNER is a prototype software library that has been developed with the aim to enable inserting ontological descriptions of activities, using the ontologies of the previous section, into an intelligent calendar application, particularly the SELFPLANNER system [Refanidis and Alexiadi, 2011]. The ONTPLANNER library transforms ontological descriptions of activities into equivalent Java objects that can be used by any software that uses Java Virtual Machine (JVM), e.g., Scala, Closure or Jython. Note however that SELFPLANNER does not support all the features of the ontologies presented in the previous section, mainly the meetings ontology, as well as some of the features of the other ontologies, whereas it supports some others. Thus, ONTPLANNER ignores any statement that is not supported by the system, whereas the ontologies, as well as ONTPLANNER, have been slightly extended to support the not yet supported SELFPLANNER features.

4.1 The ONTPLANNER Library

The ONTPLANNER library comprises three modules:

- **Ontology Parser**: This module is responsible to read the ontological descriptions of the activities and convert them into internal structures.

- **Ontology Java Model**: This is the Java object model that is used to represent internally the activity descriptions that have been read through the ontology parser. This model actually implements the model of the ontologies presented in the previous section.

- **SELFPLANNER Converter**: This is a special module, developed particularly for the integration of ONTPLANNER with SELFPLANNER. Since ONTPLANNER ontology Java model and SELFPLANNER internal model are not identical, the converter transforms Java objects from the one model to the other.
Figure 5 presents the overall architecture of the integration between ONTPLANNER and SELFPLANNER, emphasizing in the former’s internal structure. A typical use case starts with the user feeding ONTPLANNER with ontological activity descriptions, using the ontologies presented in the previous section. The ontology parser reads them and converts them to Java objects, using the ontology model module. Any additional ontology needed for the translation is loaded on the fly. Indeed, the parser module is able to extract new implicit information, using a reasoner. Finally, all the information is passed to the converter and, through it, to the intelligent calendar application.

The first two modules are general, thus they can be used with any other intelligent calendar application. The third module is application specific. Thus, in order to integrate ONTPLANNER with other intelligent calendar applications, another converter should be developed⁴.

![Figure 5. The ONTPLANNER architecture](image)

ONTPLANNER has been developed in Java version 1.8. The OWL API has been adopted to manage ontologies, whereas HermiT reasoner [Motik et al., 2009] has been employed to extract implicit knowledge. Finally, maven has been used for dependency management.

ONTPLANNER has been integrated into the latest version of the SELFPLANNER system⁵. The user has initially to copy the ontological activity descriptions in the clipboard. Then, in the main window of the system (Figure 6) he presses the “Import Ontology” button and then he pastes the activity descriptions in a text box (Figure 7).

---

⁴ The source code of ONTPLANNER, as well as the compiled library, are available at http://github.com/agnantis/ontplanner.

⁵ http://selfplanner.uom.gr
The reason why a copy/paste procedure has been adopted for importing ontological descriptions has to do with the application architecture of SELFPLANNER. Indeed, the client of the application runs as a Java applet, so security restrictions do not allow to read files from the local disk. Thus, the copy/paste option was an easy to use alternative to overcome this limitation, while allowing the user to edit the descriptions before importing them. In Figure 8, the new activities appear in the user’s activity list.

Figure 6. The main SELFPLANNER window

Figure 7. Pasting ontological activity descriptions
In order to support specific features of SELFPLANNER, some extensions to Activities and CalendarPrefs ontology were necessary. This was due to the fact that the particular ontologies had been developed as general ontologies, whereas SELFPLANNER supports special features that had not been foreseen. Particularly, two subclasses were added to class Activity: ConcreteActivity and TemplateActivity. The former represents activities already scheduled in the user's calendar, whereas the latter represents activities not yet scheduled. TemplateActivity has two subclasses, SimpleTemplateActivity and PeriodicTemplateActivity, for non-periodic and periodic activities respectively. Finally, PeriodicTemplateActivity has four subclasses, namely AnnualTemplateActivity, MonthlyTemplateActivity, WeeklyTemplateActivity and DailyTemplateActivity, thus supporting four different types of periodicity.

Particularly for SELFPLANNER, the following properties of the Activities ontology are used to describe TemplateActivity individuals:

- hasName: The name of the activity
- hasDescription: An optional description of the activity
- hasPossibleVenue: The alternative venues where the activity could take place
- hasActiveDomain: The activity's temporal domain
- hasPeriodicInterval: A set of periodicity rules, that describe when a periodic activity can be scheduled within its period.
- hasTemporalWindow: A set of temporal intervals when a non-periodic activity can be scheduled.

Furthermore, we updated the CalendarPrefs ontology so that for each preference class in the ontology, a corresponding constraint class to exist as well. The following classes have also been added to the CalendarPrefs ontology, in order to represent some special constraints or preferences types:
• Implication: Implication constraints and preferences of the form “A ⇒ B”, where A and B are activities. The semantics of an implication constraint are that in order to execute A, B has also to be executed.

• Ordering: Ordering constraint or preference of the form “A≤B”, where A and B are activities. The semantics of the ordering constraint is constraint are that when both activities A and B are scheduled for execution, all parts of activity A should be scheduled before all parts of activity B.

4.3 Examples

In this section we present two scenarios that can be described with the ONTPLANNER ontologies and can be inserted into the SELFPLANNER system.

Scenario 1: A visit to a doctor, with the visiting hours being Monday to Friday, 17:00 to 21:00.

:Doctor_Appointment rdf:type activities:WeeklyTemplateActivity, owl:NamedIndividual, owl:Thing;
activities:isInteraptable "false"^^xsd:boolean;
activities:hasName "Doctor appointment";
activities:hasPossibleVenue :Doctor_Office;
activities:hasPeriodicInterval :Week_Interval.

:Doctor_Office rdf:type activities:Office, owl:NamedIndividual, owl:Thing;
activities:hasName "Dr. Wright Office";
activities:hasLocation :Doctor_Office_location.

:Doctor_Office_location rdf:type activities:Location, owl:NamedIndividual, owl:Thing;
activities:hasLatitude 22.9202988;
activities:hasLongitude 40.6464213 .

:Week_Interval rdf:type activities:WeekInterval, owl:NamedIndividual, owl:Thing;
activities:hasTemporalDescriptor :Friday_Descriptor, :Monday_Descriptor;
activities:hasTimeInterval :Time_Interval.

:Friday_Descriptor rdf:type activities:DayDescriptor, owl:NamedIndividual, owl:Thing;
activities:hasDayOfWeek activities:FRIDAY.

:Monday_Descriptor rdf:type activities:DayDescriptor, owl:NamedIndividual, owl:Thing;
activities:hasDayOfWeek activities:MONDAY.

:Time_Interval rdf:type activities:TimeInterval, owl:NamedIndividual, owl:Thing;
activities:hasEndTime :End_Time;
activities:hasStartTime :Start_Time.

:Start_Time rdf:type activities:TimeDescriptor, owl:NamedIndividual, owl:Thing;
activities:hasMinute 0;
activities:hasHour 17 .

:End_Time rdf:type activities:TimeDescriptor, owl:NamedIndividual, owl:Thing;
activities:hasMinute 0;
activities:hasHour 21 .

Scenario 2: Writing a research article to be submitted to a journal. The writing process will happen within 3 weeks, with every writing section not exceeding 5 hours in duration (constraint), but also not being less than 2 hours (preference).
5. INFORMATION EXTRACTION FOR CONFERENCE ACTIVITIES

The third step to close the loop of the proposed holistic framework comprises the automated extraction of ontological descriptions of various activities from their textual descriptions in web pages. This is probably the most difficult step and much research has been done in the area of natural language processing. Ideally, such information should be available in ontological form; unfortunately, this is far from being the reality nowadays.

Trying to extract information for any type of activity is practically impossible. In this section, we have concentrated on automating information extraction for activities related to scientific conferences. We have defined an ontology and we have created a software module to do the task.

5.1 Information Extraction

Usually, in a scientific conference web site there is a section in the main page or in a linked one where all important dates of the conference are displayed within a table. Taking this observation into consideration, we developed a tool, called FUZZYEXTRACTOR, that is able to search for tables inside a
page, recognize interesting parts, extract them, categorize them based on the date types presented in the aforementioned ontology and, finally, export them into ontological format. A typical use case of FuzzyExtractor comprises the following steps:

i. The user provides the URL of the conference’s site.

ii. FuzzyExtractor employs its build-in parser to extract various date representations formats from arbitrary text (e.g. November 20, 2015 or 20/11/2015). Using this parser, it tries to identify the text – usually located in the first segments of the web page – that represents the dates of the main event of the conference and extract the start and end dates.

iii. The module parses the text of the page and finds all table structures. It can either select the most suitable candidate table or try to parse the data of all of them (configurable option). A simple heuristic is used to find the most suitable table: Provided a predefined list of keywords, the algorithm selects the table with the largest ratio keysNo/wordsNo, where keysNo is the number of keywords found in the table and wordsNo is the total number of words in the table.

iv. It tries to read the header of the table(s). Currently, FuzzyExtractor supports two cases: Either the header contains the type of the date (i.e., submission or notification) or there are no headers at all. In the latter case, the algorithm tries to identify the type of the date based on the text located in each line of the first column of the table.

v. The cells of the first column are parsed and their text is matched against a list of predefined keywords, in order to identify which of the various deadline types each line contains (e.g., regular paper submission deadline).

vi. Each cell in the body of the table (apart from the first column and – possibly – the first header) contains the dates we are interested in and can be extracted by the date parser. Using these dates and the data extracted from the headers and the first column of the table, the tool is able to reconstruct the information to a meaningful data structure, ready to be stored in databases, presented to the user or exported as data of the Conferences ontology.

5.2 The web application

As part of the data extraction process, we built a web application that takes advantage of both the Conferences ontology and the FuzzyExtractor tool and can be used by the users in two ways:

• As a front-end of the FuzzyExtractor that helps the user to extract date information from conferences web sites and save them as RDF data to the application’s triple-store database (Figure 9).

• As a SPARQL endpoint, where the user (through the provided SPARQL endpoint) and other machines (through the provided REST services) can impose queries related to conference data already stored in the application’s triple-store database (Figure 10).

As we can see in Figure 9, the user tries to extract information from the icaps15.icaps-conference.org web site. The application is able to identify the name of the conference, its start/end dates and various

---

6 FuzzyExtractor is available at http://coursr.uom.gr/conference
other important dates. Then the user can review the data, fill in any missing dates and, finally, save all this information to the internal triple-store database.

Due to limited space, we do not provide here the ontological description of the automatically extracted conference related activities. The interested reader can download them from: https://github.com/agnantis/uom-ontologies.

![Figure 9. The FUZZYEXTRACTOR frontend](image9.jpg)

**Figure 9.** The FUZZYEXTRACTOR frontend

![Figure 10. Querying the FUZZYEXTRACTOR internal triple-store through its SPARQL endpoint](image10.jpg)

**Figure 10.** Querying the FUZZYEXTRACTOR internal triple-store through its SPARQL endpoint

### 6. CONCLUSIONS AND FUTURE WORK

In this article we proposed a holistic framework for Intelligent Calendar Applications, based on the exploitation of ontologies. The framework comprises three parts: (a) The ontological descriptions of human daily activities, including all these details that are necessary in order to model the
computational scheduling problem; (b) an enhanced version of a deployed ICA, which accepts such ontological descriptions; and (c) an information extraction module that automatically extracts ontological descriptions of simple activities, that are ready to insert in the ICA. We have already implemented the first two parts, whereas we touched the third part by providing a solution for a particular case of it, that is, automatically extracting ontological descriptions of activities related to attending a scientific conference.

We strongly believe that ontologies have to play a crucial role in the future for the success of intelligent calendar applications. Existing calendar standards, like iCalendar, are nowadays used to describe events in a form ready to insert in traditional electronic calendars. Such descriptions can be found in web sites describing, e.g., a presentation event, or, in more dynamic cases, they are sent by email, e.g., a flight event. However, they only concern scheduled events. We envision that, in the near feature, richer ontological descriptions of not yet scheduled activities will replace static event descriptions, allowing for automated decision making about when and where to schedule these activities. We believe that the proposed framework, with the particular implementations being considered a proof of concept, is a step towards this direction.

For the future, we would like to continue working in this direction, by developing more domain activity ontologies, e.g., for professional, educational or leisure activities. Particularly, we will try to convince site owners related to any type of activity, to provide ontological descriptions of the offered activities through their sites. In order to facilitate this effort, we plan to develop add-ons to be inserted into these sites, which will facilitate the in-place ontological description of the supported activities, providing at the same time a human readable form of the related information.

7. ACKNOWLEDGMENTS

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II: Investing in knowledge society through the European Social Fund.
8. REFERENCES


Agnantis, K., Refanidis, I., 2015b. Activity Ontologies for Intelligent Calendar Applications. 7th Balkan Conference of Informatics (BCI), Craiova, Romania.


Hu, Y., Janowicz, K., 2012. Improving personal information management by integrating activities in the physical world with the semantic desktop. 20th International Conference on Advances in Geographic Information Systems, pp. 578-581. ACM.


