A demo for Smart City Operation Center

Filippos Gouidis, Giorgos Flouris, and
Dimitris Plexousakis

Institute of Computer Science, F.O.R.T.H
Heraklion, Crete, Greece
{gouidis,fgeo,dp}@ics.forth.gr

Abstract. The delivery of Public Services to citizens is developing into a challenging task, as the urban population increases and the related networks are becoming more complex and entangled. In this work, we present a GUI application for Smart City Operation Center (SMOC), a platform designed for the optimal coordination of a City's Public Agencies that is based on two eminent AI paradigms: Action Languages (AL) and Answer Set Programming (ASP). The system receives requests for services and subsequently deploys the available agents of the different services, in order to fulfil the demands in a timely manner. This paper provides the technical details behind the implementation, along with instructions and guidelines for the utilization of the application.

Keywords: Smart City Application, Answer Set Programming, Action Languages, Multi-Agent Action Coordination

1 Introduction

In the context of a modern city, the delivery of services to the citizens involves the activation of many different agencies such as the fire brigade, the police department or the health care services. Typically, the management of the actions necessary for this objective is carried out in principle by each agency separately, since it is considered that a continuous coordination is too costly and arduous. A notable exception is the handling of emergency incidents due to their critical nature.

Over the last few years, many Smart City projects have been launched with the purpose of complementing the corresponding agencies in their tasks. For example, in Amsterdam, the Smart Traffic Management Project [1] is used for the administration of traffic flow and in the EU the new emerging trend of Connected Cars [2] is expected, among others, to substantially improve ambulance response times. However, similarly with the case of the various agencies, the approach underlying behind these applications is rather specialized than holistic.

[1] http://amsterdamsmartcity.com/projects/detail/id/58/slug/smart-traffic-management

[2] http://europa.eu/rapid/press-release_ MEMO-14-105_ en.htm

Maintaining constantly a coordinated and synergistic plan, nonetheless, could be significantly more optimal for a number of reasons. Firstly, this approach is translated to a substantial cost reduction in terms of human resources and infrastructure. To start, the activities of two or more servicing organizations are often interdependent, as for example in the case of traffic regulation by the police that can facilitate enormously the other servicing units endeavouring to fulfil their tasks. Hence, a unified approach could serve better the interests of the groups involved. Last, but not least, the experience and expertise gained by this type of administration, might be proven crucial in an emergency incident such as an earthquake or an extreme weather event.

1.1 Smart City Operation Center

Towards this direction, we have developed Smart City Operation Center (SMOC), a platform for the central coordination of the various services operating in the urban environment.The main responsibility of SMOC is to manage the city services and decide how to effectively deploy the available agents of the different services, in order to provide optimum and timely assistance to citizens. The way in which the problem is represented bears strong similarities to the multiple Travelling Salesperson Problem (mTSP), a heavily studied problem in computer science and mathematics of NP-hard complexity. Nonetheless, the majority of the approaches has domain-dependant performance and the extension of the representation of the problem is far from trivial [1].

Our main objective behind this effort aims at providing an easy-to-use tool that could cope with a challenging practical problem. In order to develop the platform, we utilized two eminent AI paradigms: Action Languages (AL) and Answer Set Programming (ASP), considering that the declarative and highly expressive nature of these formalisms is well suited for the addressing of the problem. In addition, the aforementioned characteristics facilitate significantly the development of solutions that are easily extended and can be used in problems with similar settings. It should be stated that the version presented in the context of this work is not final and is amenable to significant improvement.

2 Background

2.1 Event Calculus

Action theories are logical languages for reasoning about the dynamics of changing worlds, having played a pivotal role in the development of non-monotonic logics and in formalisms to represent knowledge. The EC [2, 3] is a narrative-based many-sorted first-order language for reasoning about action and change, where the sort $\mathcal{E}$ of *events* (or actions) indicates changes in the environment, the sort $\mathcal{F}$ of *fluents* denotes time-varying properties and the sort $\mathcal{T}$ of *time-points* is used to implement a linear time structure. The calculus implements the *principle of inertia* for fluents, which captures the property that things tend

to persist over time unless affected by some event, and applies the technique of *circumscription* to solve the frame problem and support default reasoning.

Different dialects have been proposed over the years; in this study, we axiomatize our domains based on the Discrete-time Event Calculus (DEC) [4] and the recently proposed Functional Event Calculus (FEC) [5]. DEC defines a set of predicates to express which fluents hold when ($HoldsAt \subseteq \mathcal{F} \times \mathcal{T}$), which events happen ($Happens \subseteq \mathcal{E} \times \mathcal{T}$), what their effects are ($Initiates$, $Terminates$, $Releases \subseteq \mathcal{E} \times \mathcal{F} \times \mathcal{T}$) and whether a fluent is subject to the law of inertia or released from it ($ReleasedAt \subseteq \mathcal{F} \times \mathcal{T}$).

FEC, on the other hand, generalizes the EC to include non-binary (i.e. non-truth-valued) fluents taking values from the sort $\mathcal{V}$. In accordance to DEC, the key predicates and functions are $Happens \subseteq \mathcal{E} \times \mathcal{T}$, $ValueOf : \mathcal{F} \times \mathcal{T} \rightarrow \mathcal{V}$, $CausesValue \subseteq \mathcal{E} \times \mathcal{F} \times \mathcal{V} \times \mathcal{T}$, $PossVal \subseteq \mathcal{F} \times \mathcal{V}$. Non-determinism and triggered actions are supported by both formalisms. Along with the set of domain-independent rules that axiomatize the notions of inertia, causality and effect, the execution of reasoning tasks is performed with a set of domain-dependent axioms expressing the dynamics of the world.

2.2 Answer Set Programming

Answer set programming (ASP) is a recently developed programming technique combining declarativeness, modularity and expressiveness. It is founded on logic programming answer set semantics, which drive the computation of stable models (answer sets). The procedure followed by most of ASP solvers is an enhanced version of the DPLL algorithm [6]. ASP is gaining increasing popularity due to its ability to combine an expressive, non-complex language over powerful solvers.

An Answer Set Program is a set of rules of the form:

Rule: A_0 `:-` L_1 `,` $\ldots$ `,`L_{k-1},not L_k,...,not L_n `.`

where L_j are atoms and *not* represents negation-as-failure. The set of literals $\{L_1, \ldots, L_n\}$ are called the body of the rule and A_0 the head. Intuitively, the head of a rule has to be true whenever all its body literals are true in the following sense: a non negated literal L_i is true if it has a derivation and a negated one, `not` L_i, is true if the atom L_i does not have one. According to stable model semantics, only atoms appearing in some head can appear in answer sets. Furthermore, derivations have to be acyclic, a feature that is important to model reachability. Rules with an empty body are called facts and their head is unconditionally true, i.e., it appears in all answer sets. Rules with an empty head are called integrity constraints and are used to reject answer set candidates.

ASP has already proven its potential in expressiveness in comparison to other declarative approaches, enabling the representation of phenomena for commonsense and nonmonotonic reasoning ([7, 8]), while its solvers outperform satisfiability-based and constraint-programming solvers in many domains [9, 10]. The success of ASP is demonstrated in a wide variety of fields that spans from hardware design and phylogenetic inference to the Semantic Web. In this study, we use the most recent ASP implementation developed by Potsdam Answer Set Solving Collection (Potassco), named Clingo [11]. It combines the grounder Gringo

and solver Clasp and encompasses many utilities, such as detailed tuning of grounding and solving, utilization of useful built-in functions and also the ability to integrate scripts written in Lua and Python languages, which provides significant flexibility to developers.

3 Technical Details

In order to address the $\mathcal{SCOC}$ setting, planning with a combination of features, such as temporal and causal constraints, was necessary. The problem incorporates characteristics of simpler frameworks, such as variations of the Traveling Salesman Problem, distance graphs and temporal reasoning with precedence ordering. Furthermore, it extends them in various ways. In this section, we describe how the EC and ASP can be applied to approach representational and practical issues related to the problem of $\mathcal{SCOC}$, revealing strong and weak points of each.

3.1 Representation

Representing $\mathcal{SCOC}$ with Event Calculus Axiomatizations The $\mathcal{SCOC}$ planning problem formulates a demanding domain; while RAC theories are well suited to express some of its aspects, others prove to be more challenging, as we discuss next. We chose the EC as the specification language to describe the domain, not only due to its ability to model a multitude of commonsense phenomena, but also because of the availability of tools that can support our reasoning tasks.

In order to model the setting both FEC and DEC have been used. However, even though both theories are comparable for the given domain, we will focus in this subsection on FEC, whose ability to model functional fluents offers greater flexibility. Namely, as it is already elaborated in the Background section, FEC provides the possibility of using non-binary fluents. Moreover, in contrast to DEC, FEC can model non-determinism by the usage of disjunction ([5]), a feature that could be exploited in the context of a future expansion of the platform aiming to support uncertainty.

We picture the smart city as a graph, whose edges have weights that may change dynamically as a result of occurring events. To simplify the presentation we assume one service and one agent type; the axiomatization can trivially be extended to the more general case, by simply using a different graph per agent type/service. In compliance with the notation introduced in the previous sections, let $ag, ag_1, ...$ denote variables of the $\mathcal{AG}$ sort, $l, l_1, ...$ variables of the $\mathcal{E}$ sort, while variables $num, num_1, ...$ denote positive reals[3].

The following domain closure axioms define all fluents and actions needed to model $\mathcal{SCOC}$, in order to reason about the state of agents (i.e., being at a

[3] All variables in formulae are implicitly universally quantified, unless stated otherwise. Moreover, we assume $\mathcal{E} \supseteq \mathcal{AC} \cup \mathcal{EV}$, i.e., events axiomatized by the Event Calculus refer both to agent actions and triggered events.

location or moving), the state of locations (i.e., served or not) and the distance traveled:

$$f = At(ag) \vee f = Moving(ag) \vee f = RemDist(ag, l_1, l_2) \vee f = Step(l_1, l_2) \vee f = Served(l).$$
$$e = Departs(ag, l_1, l_2) \vee e = Arrives(ag, l) \vee UpdateRemDist(ag, l_1, l_2, num).$$

Fluent *Step* denotes how much distance the agent can cover in one timepoint along the edge (l_1, l_2), while *RemDist* captures the distance that remains to be traveled. *Step* is subject to change, as it depends on the state of the world. We further assume uniqueness of names axioms for actions and fluents. The possible values for these fluents are appropriated restricted:

$$PossVal(At(ag), l).\ PossVal(Step(l_1, l_2), num).\ PossVal(RemDist(ag, l_1, l_2), num).$$
$$PossVal(Moving(ag), v) \equiv v = True \vee v = False.$$
$$PossVal(Served(l), v) \equiv v = True \vee v = False.$$

Certain predicates are also defined. For instance, $Connected(l_1, l_2, num)$ denotes edges and the corresponding distance between locations.

In comparison to other action theories, the explicit representation of time inside EC predicates facilitates the developer in expressing complex temporal expressions, necessary in $\mathcal{SCOC}$ to model for instance *actions with durations*, such as traveling for a given amount of time. Moreover, the calculus has established solutions to the frame, ramification and qualification problems, relieving the developer from the tedious work of explicitly writing frame or minimization axioms. Many aspects of the domain can easily be described by axioms expressing *context-dependent effects of actions*, *action preconditions* and *state constraints*, with existentially quantified variables whenever needed:

$$CausesValue(Arrives(ag, l), at(ag), l, t) \leftarrow ValueOf(Moving(ag), t) = True.$$
$$Happens(Departs(ag, l_1, l_2), t) \rightarrow ValueOf(At(ag), t) = l_1.$$
$$ValueOf(At(ag), t) = l_1 \wedge ValueOf(At(ag), t) = l_2 \rightarrow l_1 = l_2.$$
$$Happens(Departs(ag, l_1, l_2), t) \rightarrow \exists num Connected(l_1, l_2, num).$$

In order to handle metric distances between locations or calculate traveled distances, $\mathcal{SCOC}$ calls for extensive use of *numerical operations* to be incorporated in the domain description. For instance, we have to recalculate the *Step* fluent every time certain agent actions affect it, such as when some agent arrives at, serves or leaves a particular location. We use the predicate $AffectsCost(ag, l, l_1, l_2, num, v)$ to model different variations of the *costChange* event introduced in Section 3. For instance, when $v = 1$ (resp. $v = 2$, $v = 3$) *AffectsCost* denotes that the cost of traveling from l_1 to l_2 becomes *num* when agent ag arrives at (resp. is present at, departs from) location l. The following axiom models the case when $v = 3$ (the rest are similarly defined):

$$CausesValue(Departs(ag, l), Step(l_1, l_2), (num_1/num), t) \leftarrow$$
$$[Connected(l_1, l_2, num_1) \wedge ValueOf(At(ag), t) = l \wedge affectsSpeed(ag, l, l_1, l_2, num, 3)].$$

Despite the simplicity of such axioms, the introduction of numerical variables leads to an explosion of grounded terms having a tremendous impact on performance, calling for special measures to be adopted.

Finally, *triggered events*, i.e., events that occur when the world is in a particular state, are a significant leverage in modeling real-world domains [12]. In our case, we use triggered events to keep track of the distance that an agent needs to travel before reaching the destination location:

$$Happens(UpdateRemDist(ag, l_1, l_2, (num_1 - num_2)), t) \leftarrow ValueOf(Moving(ag), t) = True \wedge$$
$$ValueOf(RemDist(ag, l_1, l_2), t) = num_1 \wedge ValueOf(Step(l_1, l_2), t - 1) = num_2.$$

Note that, in contrast to most benchmark problems in action theories, in our case the duration of certain actions, such as the agent's travel is not known beforehand, but needs to be calculated on-the-fly. Our modeling adopts the simple solution of recalculating the remaining distance at every timepoint according to the distance the agent has traveled in the previous timepoint (notice that $Step$ refers to $t-1$ in the previous axiom). A variation of this problem with static edge weights has also been implemented to show the difference in performance when action duration is known *a priori*. ASP constructs can provide radical solutions, as we argue in the next section.

Finally, the axiomatization needs also to include the description of the initial state, specifying the location and state of all agents, as well as the goal state, i.e., $ValueOf(Served(n), T_{opt}) = True$ for some timepoint T_{opt}. Note that since the problem we solve is a planning problem, we do not specify completion of the $Happens$ predicate, letting the reasoner produce all combinations of event occurrences that can lead to the satisfaction of the goal state. Of course, T_{opt} is not known beforehand.

Representing $\mathcal{SCOC}$ in Answer Set Programming This subsection describes an alternative modeling that uses ASP as specification language for describing $\mathcal{SCOC}$, aimed at exploiting the potential of state-of-the-art ASP solvers. Given the structure of the problem, we based our representation on standard methods found in the literature for encoding graph traversal, as given for instance in [13]. Due to the dynamic nature of the problem, we extended the methodology with a treatment of time. In this way, atoms representing dynamic attributes contain a variable accounting for time. As before, we simplify the setting and assume only one agency and every node of the graph has to be visited at least by one of the agents.

To accommodate planning, Clingo offers a special functionality where reasoning progresses incrementally. Specifically, a special variable, which denotes timepoints in our case, is acting as a place-holder that increases by a constant step number to perform grounding and solving in consecutive steps, until an answer set satisfying the goal state is found. To accomplish this, the program is divided into 3 independent parts: the basic, the cumulative and the volatile part. The former contains the definitions that are used throughout the execution, as well as the initial state, while the latter specifies the goal condition. The cumulative part, on the other hand, incorporates all rules and constraints that have to be grounded every time the reasoning progresses by one step.

The state of the graph is specified by predicates, such as `in(Ag1,Nd1,1)` and `edge(Nd1,Nd2,W,1)` contained in the basic part. In order to represent the displacement of the agents, rules able to express cardinality constraints are used:

$$0\{on\_the\_road(AG, X, Y, C, t) : edge(X, Y, C, t)\}1 :- agent(AG), in(AG, X, t).$$

This rule states that an agent located at a node at a certain time-point can initiate *at most* one movement and this movement should have as destination a node that is connected (i.e., `edge`) to the node that is currently located at.

Such rules give significant leverage to the developer, offering a compact way to introduce various types of restrictions.

In order to avoid overloading the grounded terms introduced in the Event Calculus encodings when numerical operations are in place, we embedded in our ASP encodings *external predicates*, a special functionality offered by Clingo. The truth value of these predicates can be decided by scripts written in the Lua language, without requiring grounding or disturbing execution during reasoning. Such an external predicate is $@new\_cost$ appearing in the following rule:

$$on\_the\_road(AG, X, Y, C\_NEW, t) : -on\_the\_road(AG, X, Y, C\_OLD, t-1), 0 < C\_OLD,$$
$$e(X, Y, W\_OLD, t-1), e(X, Y, W\_NEW, t), C\_NEW = @new\_cost(W\_NEW, W\_OLD, C\_OLD, t).$$

The rule calculates the remaining distance for agents on the move. The performance gains obtained when executing such computationally demanding tasks in parallel with reasoning is depicted in the evaluation discussed in Section 5.

While the built-in constructs described before offer enhanced functionalities to support declarative reasoning, certain aspects of $\mathcal{SCOC}$ were not handled as conveniently as with the EC encodings. The representation of inertia, which had to be explicitly defined in all time-dependent rules, and the treatment of time in general, are characteristic examples:

$$-edge(X, Y, C2, t) : -edge(X, Y, C, t), edge(X, Y, C2, t-1), C2! = C.$$
$$edge(X, Y, C2, t) : -edge(X, Y, C2, t-1), not\ -edge(X, Y, C2, t).$$

These rules model the weight of edges at each timepoint, having the law of inertia explicitly expressed. The first rule implies that if for two consecutive time moments an edge has different weights, the earlier value must become obsolete the next time moment, while the second rule indicates that if an edge's value has not become obsolete, it is conserved for the next time moment ("$-$" denotes strict negation). Similar behavior has to be designed for other atoms, whose value may change over time. The ease of accommodating such phenomena with the EC and their direct application to new features with minifemum effort, often referred to as elaboration tolerance, becomes easily evident.

Finally, the constraint expressing the goal condition, i.e., whether all nodes have been visited, is expressed as follows and added in the volatile part:

$$: -not\ reached(X, t), node(X), query(t).$$

We additionally made use of Clingo's built-in function *minimize*, in order to achieve a second-level of optimization:

$$\#minimize\{C : on\_the\_road(AG, X, Y, C, t)\}$$

With this expression we can define a secondary criterion to classify optimal plans, when more than one are found. Specifically, we choose the one with minimum distances traveled by all agents, denoted by variable C. Special treatment of such expressions allows the Clingo solver to calculate solutions effectively.

3.2 Implementation

Recent progress regarding the generalization of the definition of stable model semantics used in ASP [14] has opened the way for highly expressive formalisms to

be reformulated in ASP encodings and exploit the several efficient implementations that are currently available. Specifically, the precise characterization of the correspondence between stable models and circumscription used by many theories for reasoning about action and change, such as the Situation and the EC, has permitted the reformulation of the latter in ASP. For that purpose, we used the F2LP[4] tool to transform our circumscriptive DEC-based axiomatization into a logic program that can be executed with ASP reasoners. This is important since not all first- order formulae can be transformed into the clausal form used in ASP solvers while preserving stable models. F2LP applies the translation developed in [15] that guarantees that the ASP encoding created as output is equivalent to the circumscription-based axiomatization. For example the following DEC axiom:

$$happens(departs(Ag, Nd1, Nd2), T) \rightarrow$$
$$terminates(departs(Ag, Nd1, Nd2), at(Ag, Nd1), T).$$

is transformed via F2LP into the next ASP rule:

$$terminates(departs(Ag, Nd1, Nd2), at(Ag, Nd1), T) : -$$
$$happens(departs(Ag, Nd1, Nd2), T), agent(Ag), node(Nd1), node(Nd2), time(T).$$

Although the FEC-based axiomatization can also facilely become input to F2LP, we opted to utilize the dedicated reasoner and encoding style for FEC theories developed in [16]. The major advantage that this tool offers lies in its capacity to support reasoning with highly expressive classes of problems with minimum effort on the developers side. Namely, it can execute the epistemic extension of FEC [5], which we plan to integrate in future variations of the SCOC setting, when for example some of the agents locations will not be known initially.

In order to compare these different implementations in terms of efficiency and speed, a number of experiments were conducted on random generated clique graphs. The main findings were the following: 1) in general, none of the implementations could scale well in large cliques ,2) the purely ASP-based implementation scored significantly better than the hybrid ones, 3) the major drawback of EC was its incompetence to deal with the numerical operations and 4) the grounding phase is far more time-consuming than the solving phase. For more details regarding the experiments the reader is referred to [17]

At first glance, based on the previous results, it can be argued that the utilization of EC in the current framework of the problem is unnecessary and, hence, only the ASP implementation should be applied. Nevertheless, at it has already been stated, the platform is still under development and the wide experimentation which we plan to conduct regarding the EC encodings, might lead to enhanced performance. Furthermore, the unique functionalities that EC offers could be proved particular useful for the extension of the problem. Summarizing, we suggest that the user concerned only in performance issues should use the purely ASP encoding. However, we opted to offer, in the context of the current version, all the different options in order for a more direct comparison to be possible.

[4] F2LP website (last accessed 7/7/2014): `http://reasoning.eas.asu.edu/f2lp/`

4 User Guide

4.1 Requirements

Currently, the application can be executed only in a Linux environment. However, we plan to release a version that supports the family of Windows OS in the near future. In order for the application to be executed, the following three components are required: 1) Java Runtime Environment (JRE)[5] , 2) a 2.7 version or higher of the Python language[6] and 3) version 5.1 or higher of the Lua language [7] .

The necessary files for the deployment of the application are located in the following url: `http://www.ics.forth.gr/isl/MACPDRA/Demo/Demo.zip` . Extract the compressed folder Demo.zip and then launch the Demo.jar file with JRE. Additionally two tutorial videos can be found in the following urls: `http://www.ics.forth.gr/isl/MACPDRA/Demo/video1.avi` and `http://www.ics.forth.gr/isl/MACPDRA/Demo/video2.avi` . Please note that for the seamless execution of the application none of the files included in the uncompressed folder should be deleted. Also, to prevent any mishaps, ensure that the permissions of the files enable their execution.

It should be stated that some of the features of the problem are not yet supported by the application, although the corresponding ASP and AL encodings address them. Namely, there is no difference in the type of the different agents, i.e. they all deliver the same services, and the edges of the graphs are static, that is there is no dynamic interdependency among the agents. Nonetheless, soon these aspects will be incorporated in the application.

4.2 Basic Commands

The application consists of two sub-windows. The left one is dedicated to the design of the graph that represents the locations of the city, whilst in the right the paths that the agents have to travel to provide the servicing are depicted.

There are two options: the user can either use the strictly ASP implementation or the implementation that is a combination of ASP and AL. By selecting the former, the user can select the vertices of the graph that require servicing. On the contrary, in the latter case it is assumed that all vertices should be visited by the agents. In order to select between these two options, the user must use the combo button that is located below the left sub-window.

In order to create a new vertex, the user should right-click in an empty space in the left sub-window. Consequently a name for the new vertex should be given. Note that each vertex should bear a unique name.

Similarly, to create a new edge between two vertices the user must right-click on one of the two vertices and then drag the mouse to the other. Then, the weight of the edge should be given in the pop-up dialog that appears. The value should be a positive integer.

[5] http://www.oracle.com/technetwork/java/javase/overview/index.html
[6] http://www.python.org/
[7] http://www.lua.org/

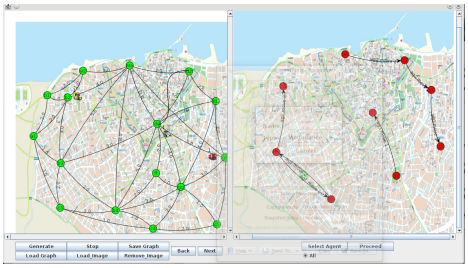

Fig. 1: Application's main window

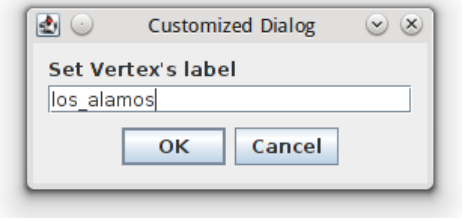

Fig. 2: Creating a vertex

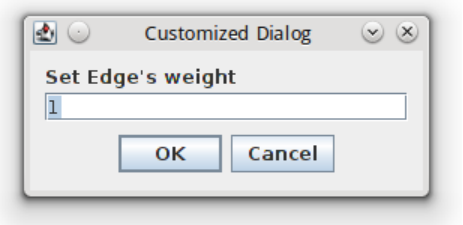

Fig. 3: Creating an edge

To add a new agent in a vertex, the user must right-click the vertex. In the dialog that appears the type and the name of the agent should be given. In the same way, the user can remove an agent from a vertex by selecting the option 'Remove Agent'.

To indicate that a vertex requires servicing, the user must right-click the vertex. After the service have been added, the color of the vertex turns from green to red. In order to remove a service from a location, the user must right-click the vertex and select 'Delete Service'.

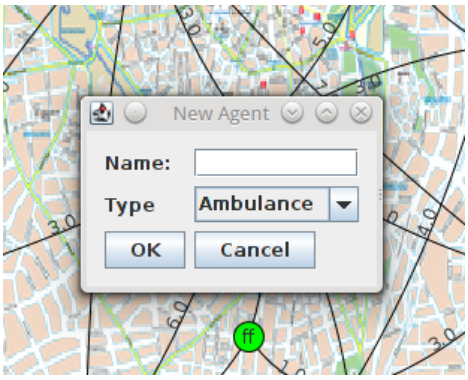

Fig. 4: Adding an agent

Fig. 5: Designing a graph

To add an image as a canvas, the user must click the button 'Load Image' that is located under the right sub-window. The image can be removed by clicking the button 'Remove Image'. At any time, the user can save a graph in an XML

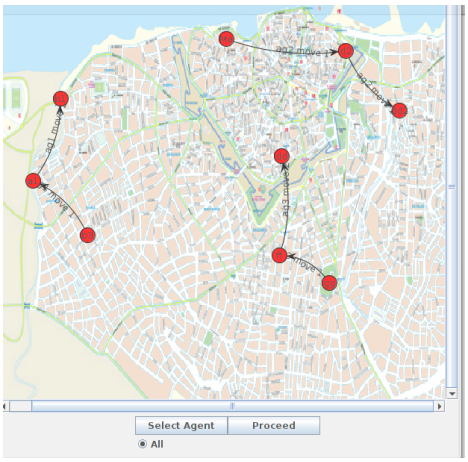

Fig. 6: A generated plan indicating the paths of the agents

file by using the button 'Save Graph'. Correspondingly, the user can load any previously saved graphs by using the button 'Load Graph'.

After the graph has been designed, the user can generate the plan for the agents by clicking the button 'Generate' that is located under the right sub-window. By clicking the button 'Proceed', the agents advance according to the previously generated plan.

5 Conclusions

In this paper, we presented a smart city application that relies on the formalisms of AC and ASP. While the investigation behind its implementation builds on the latest theoretical and applied advancements in both fields, its user-friendly interface enables people less or not familiar with these theories to harness its potential and utilize it in the context of a highly demanding domain: a Smart City environment. Conversely, by exploiting the direct and swift visualization rendering that this tool provides, we could be aided significantly in our research by gaining insights about the merits and the drawbacks of the aforementioned paradigms.

Before concluding, we would like to stress, once again, that the platform is currently under development and, therefore, does not lack deficiencies, the most important of which having to do with scalability and computational issues. Besides, as SCOC will expand, we plan to include more features such as unpredictable events, durational and prioritized tasks. This way, we aspire toward

developing more efficient, real-world implementations for SCOC. Moreover, the GUI can be further improved and simplified, facilitating, thus, to a larger degree the inexperienced user. Nonetheless, despite the enhancements and adjustments that might be required, it is our belief that SCOC already constitutes a reliable tool that can be used for both practical and theoretical applications.

References

1. Kumar, T.S., Cirillo, M., Koenig, S.: On the traveling salesman problem with simple temporal constraints. In: 10th Symposium of Abstraction, Reformulation, and Approximation(SARA'13). (2013) 73–79
2. Kowalski, R., Sergot, M.: A Logic-based Calculus of Events. New Generation Computing **4**(1) (1986) 67–95
3. Miller, R., Shanahan, M.: Some alternative formulations of the event calculus. In: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II, London, UK, Springer-Verlag (2002) 452–490
4. Mueller, E.: Commonsense Reasoning. 1st edn. Morgan Kaufmann (2006)
5. Miller, R., Morgenstern, L., Patkos, T.: Reasoning about knowledge and action in an epistemic event calculus. In: 11th International Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense'13). (2013)
6. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM (JACM) **7**(3) (1960) 201–215
7. Eiter, T., Ianni, G., Krennwallner, T.: Reasoning web. semantic technologies for information systems. (2009) 40–110
8. Coban, E., Erdem, E., Ture, F.: Comparing ASP, CP, ILP on two Challenging Applications: Wire Routing and Haplotype Inference. In: Proc. of the 2nd International Workshop on Logic and Search (LaSh 2008). (2008)
9. Kim, T.W., Lee, J., Palla, R.: Circumscriptive event calculus as answer set programming. In: 21st International Joint Conference on Artificial Intelligence (IJCAI-09). (2009) 823–829
10. Celik, M., Erdogan, H., Tahaoglu, F., Uras, T., Erdem, E.: Comparing ASP and CP on four grid puzzles. In: Proc. of the 16th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'09). (2009)
11. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The potsdam answer set solving collection. Ai Communications **24**(2) (2011) 107–124
12. Tran, N., Baral, C.: Reasoning about Triggered Actions in AnsProlog and Its Application to Molecular Interactions in Cells. In: 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR2004). (2004) 554–564
13. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers (2012)
14. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence **175**(1) (2011) 236–263
15. Lee, J., Palla, R.: Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. Journal of Artificial Intelligence Research **43**(1) (2012) 571–620

16. Ma, J., Miller, R., Morgenstern, L., Patkos, T.: An epistemic event calculus for asp-based reasoning about knowledge of the past, present and future. In: Proc. of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19). (2013)
17. Gouidis, F., Patkos, T., Flouris, G., Plexousakis, D.: Declarative reasoning approaches for agent coordination. In: Artificial Intelligence: Methods and Applications. Springer (2014) 489–503