University of Groningen

Automated Service Composition Using AI Planning and Beyond

Medema, Michel; Kaldeli, Eirini; Lazovik, Alexander

Published in:

Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future: Essays Dedicated to Michael Papazoglou on the Occasion of His 65th Birthday and His Retirement

DOI:

[10.1007/978-3-030-73203-5_2](https://doi.org/10.1007/978-3-030-73203-5_2)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version

Publisher's PDF, also known as Version of record

Publication date:

2021

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Medema, M., Kaldeli, E., & Lazovik, A. (2021). Automated Service Composition Using AI Planning and Beyond. In M. Aiello, A. Bouguettaya, D. A. Tamburri, & W-J. van den Heuvel (Eds.), *Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future: Essays Dedicated to Michael Papazoglou on the Occasion of His 65th Birthday and His Retirement* (pp. 16-32). (Lecture Notes in Computer Science; Vol. 12521). Springer International Publishing. https://doi.org/10.1007/978-3-030-73203-5_2

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Automated Service Composition Using AI Planning and Beyond

Michel Medema¹(✉) , Eirini Kaldeli², and Alexander Lazovik¹

¹ University of Groningen, Groningen, Netherlands
{m.medema,a.lazovik}@rug.nl

² National Technical University of Athens, Athens, Greece
ekaldeli@image.ece.ntua.gr

Abstract. Automated Service Composition is one of the “grand challenges” in the area of Service-Oriented Computing. Mike Papazoglou was not only one of the first researchers who identified the importance of the problem, but was also one of the first proposers of formulating it as an AI planning problem. Unfortunately, classical planning algorithms were not sufficient and a number of extensions were needed, e.g., to support extended (rich) goal languages to capture the user intentions, to plan under uncertainty caused by the non-deterministic nature of services; issues that were formulated (and, partially addressed) by Mike, being one of his key contributions to the service community.

In this chapter, we look at the development of the original vision of automated service composition as AI planning, going from planning with extended (rich) goals, further developing into composition under uncertainty, extending it to other domains (and reformulating the service composition as composition of sensors and actuators), and then showing possible alternative techniques for highly scalable service composition at the expense of the richness of the domain representation.

Keywords: Automated service composition · AI planning · Constraint satisfaction · Internet of Things

1 Introduction

Service-Oriented Architectures (SOA) are a modern approach for developing software, where individual, loosely-coupled software components (called services) are accessed via platform-independent APIs [1]. The developed application is then defined by its services and a separate wiring logic for combining them [10]. Often such logic is also defined in a platform-independent language, such as BPEL or BPMN [26]. The process code is typically written in the same way as traditional software—by manually typing code. However, considering the high-level nature of the process languages, such abstractions are good candidates for

This work has been partially sponsored by EU H2020 FIRST project, Grant No. 734599, FIRST: vF Interoperation suppoRting buSiness innovaTion.

© Springer Nature Switzerland AG 2021

M. Aiello et al. (Eds.): Papazoglou Festschrift, LNCS 12521, pp. 16–32, 2021.

https://doi.org/10.1007/978-3-030-73203-5_2

a computer-aided composition, where users define “important” properties of the desired composition, and the rest—actual process code generation—can be done in an automated way. Mike Papazoglou was one of the pioneers in emphasising the importance of automated service composition [28] and AI Planning [2, 27] being one of the possible solutions for realisation of the idea in practice.

Several approaches inspired by work in the AI planning community have been proposed in order to move towards compositions characterised by a higher degree of automation and adaptability to different user needs and changing environmental conditions, e.g., [6, 24, 29, 31]. Planning has been described as the process of “choosing and organising actions by anticipating their expected outcomes”, with the aim to achieve some well-defined goal [15]. Certain analogies with the problem of service composition can be drawn: the functionalities offered by services can be modelled as *actions*, with preconditions/inputs and effects/outputs, and the goal reflects the request of the user.

In this chapter, we provide an overview of one of the research lines that originated from automated service composition. First, we look at automated planning for extended goals under uncertainty [19], then we apply it to the domain of smart homes [20] and smart offices [14], and, finally, extend the underlying methods of constraint satisfaction to exploit the nature of the problem structure typical for automated service composition in the domain of large scale smart environments. The methods and techniques have also transformed over the years. Initially, services and their operations are uniform and represented as actions in the AI planning problem. To model uncertainty, some service operations are modelled as non-deterministic actions, with unknown outcome. To represent active domain exploration, some services are defined as “knowledge-gathering”, which are then used to collect the information about the world state. The “knowledge gathering” and “planning” are then executed repeatedly, leading to the interleaving of planning and execution. In the domain of smart homes and smart offices, a clear distinction is made between knowledge gathering (sensors) and actions altering the world (actuators). The values of sensors can be read at any time and do not require a separate planning phase. For domains where dependencies between services are not as important, the service composition can be simplified, and planning can concentrate on identifying a small set of actions based on changes in the environment that are then all applied simultaneously during execution. While simplifying the service composition reduces the richness of the domain description language, it allows for automated service composition in large scale domains consisting of many thousands of services (which is typical for smart building and smart city applications).

The rest of the chapter is organised as follows. Section 2 discusses the use of planning for automated service composition. Section 3 presents an approach to automated service composition with extended goals and uncertainty based on encoding a planning problem as a constraint satisfaction problem. An example application of automated service composition in a smart home is provided in Sect. 4. In Sect. 5, we show a different approach to automated service composition, where the services are explicitly modelled as either sensors or actuators, and relations between different services are defined by the constraint rules. Finally, the conclusions are drawn in Sect. 6.

2 Planning for Automated Service Composition

2.1 Expressing User Goals

There are several ways in which the goal can be formulated, and different goal languages vary in the expressivity they support and the flexibility they allow with respect to the possible sequences of actions that can lead to the desired outcome. Different approaches have been adopted in this respect: some build on goal specifications that express in a declarative way the objective to be achieved (e.g. [29,30]) and others prescribe some form of a procedural template, either in the form of Hierarchical Task Network methods, e.g., [3], as a Golog program, e.g., [31], or as a target state automaton, e.g., [8]. The main shortcoming of the latter line of work is that by dictating the possible connections between the Web service operations, they restrict the applicability of the domain to only some predefined set of anticipated user behaviours. Unlike these approaches, a declarative goal specifies *what* has to be achieved and leaves it to the automated planner to compute the *how*. Such a domain-independent approach fits well with the loosely-coupled nature of services that can be combined in many different ways to serve a variety of objectives. The idea is to maintain a generic and modular repository that comprises a number of atomic service operations, from booking flights to arranging appointments with a doctor, and leave it to the planner to automatically generate the composition which satisfies the desired goal and abides to certain constraints that ensure the prevention of undesirable situations. At the same time, the goal language should be expressive enough to capture complex goals that go beyond the mere statement of properties that should hold in the final state, being able to support conditions over state traversals, maintainability properties, and distinguishing between the desire to observe the environment and the desire to change it. What is needed is a rich declarative language for expressing extended goals, without requiring from the user to be aware of the particularities and underlying inter-dependencies of the available services.

2.2 Dealing with Uncertainty

Service environments are characterised by a high degree of uncertainty in multiple levels. Firstly, there is uncertainty about the initial state, i.e., certain variables may be unknown and certain services have to be invoked in order to get to know their value. In fact, marketplaces consisting of services publicly available on the Web are usually dominated by services that are data sources [31], which in a planning context are modelled as sensing operators (also referred to as knowledge-gathering or observational operators). A successful plan may be conditioned on the outcomes of such actions, e.g., the user may want to go ahead with buying a book from [amazon.com](https://www.amazon.com) if it costs less than a maximum price. In a domain-independent setting, the planning agent is required to plan for sensing; the planner should be able to identify which knowledge it lacks for satisfying the goal, and reason about how to find it. The planner should also proactively

take care of the data flow entailed by a service composition, i.e., the way in which values returned by observation actions are used by subsequent actions. For example, a user may want to mail a parcel to someone whose address he does not already know. The plan should thus first consult a white pages service, and then give the order for posting by passing the right address value to the respective input parameter. For data intensive service domains, determining the parameters of an action can be just as difficult as determining which actions belong to the plan.

Besides the uncertainty due to the different possible outputs of sensing actions, non-determinism about the results of an action's execution may also stem from *runtime* contingencies. Service environments are unpredictable and a service invocation may behave in unexpected ways: it may return a failure, not respond at all, or even act in a way different from the one prescribed by its description. To add to the volatility of service environments, the activity of exogenous agents may also change the dynamics of the domain in unintentional ways. For example, considering a smart home environment and a partially executed composition which involves the steering of a Hoover, if an external actor puts an obstacle in the Hoover's way in the middle of execution, the composition may fail unless decisions about how to move are revised. Moreover, in many service environments, dynamicity also applies to the availability of information and services; e.g., the services offered by a mobile phone may appear and disappear depending on the location of the phone. Thus, the problem of uncertainty is directly related to the interaction between planning and execution. With a few notable exceptions [3, 7, 22], context dynamicity has largely been overlooked by existing planning approaches to WSC.

3 The RuGPlanner

The RuGPlanner [19] has been proposed as a solution to address the special requirements put forward by service domains and has been used in different domains, including Web services [18], the environment of a smart home [9, 20, 21], and for the dynamic repair of business processes [4, 5]. The RuGPlanner's underlying approach is based on translating the domain and the goal into a Constraint Satisfaction Problem (CSP) and applying a state of the art constraint solver to compute a solution-plan. Planning as CSP fits well with many aspects that are of particular concern for service domains. CSP formalisms are expressive, since constraints in the context of a multi-valued encoding allow us to naturally go beyond logical formulas, and use arithmetic formulas in preconditions and effects without sacrificing efficiency. Describing the world in terms of variables which may range over a domain of possible values is particularly convenient for representing "under-defined" states: the range of the allowed values depends on the current knowledge of the agent and is narrowed down by the application of action effects. With such an encoding, states at which variables are not restricted to a specific value, represent sets of states, thus naturally encompassing uncertainty. Moreover, such an encoding is well suited to most standard Web service

description languages, such as WSDL¹, which are based on state variables rather than predicates; complex goals can also be expressed in the form of constraints. Finally, plans can include parallel actions, which are particularly useful at execution time.

To deal with uncertainty and the possible mismatch between offline planning and actual execution results, our proposal relies on continual planning: upcoming plan steps anticipated offline can be revised as execution proceeds in the face of inconsistencies that stem either from the newly sensed information, erroneous service behaviour or the actions of exogenous agents. We describe this alternating sequence of offline planning and online execution as orchestration, a well-known concept in the Service-Oriented Computing community which denotes how the execution of a composition is managed. In a service infrastructure, the orchestration engine is a central coordinator which interacts with the component Web services in accordance with the composite process specifications. We reserve a similar role for an orchestration component that interacts with the environment, informs the planner about the data it has collected, and decides when to switch from planning to execution and vice versa. Depending on the situation, re-using parts of the existing plan may speed up the process of plan revision. Moreover, a non-blocking strategy is adopted with respect to waiting for the response of sensing actions, so that the framework can go on with the planning and execution of actions that do not depend on the expected response. Continual planning can be nicely incorporated into CSPs. In fact, dynamic constraint solving allows for the efficient addition and removal of constraints. This enables the planner to constantly incorporate new facts about the environment or remove obsolete ones, check for possible inconsistencies, and react accordingly.

In general, under conditions of uncertainty about the initial state, the search space is no longer the set of states of the domain, but its power set. The planner can resort to sensing operations to acquire all the knowledge it misses and which is necessary to fulfil a goal. Sensing operations return exactly one outcome amongst a (possibly very large) set of deterministic choices. In that respect, a plan computed online represents a traversal between sets of states rather than complete descriptions of states, and has only the potential to achieve the goal, if there is some state sequence that could arise from the plan's execution and satisfies the goal. Such a plan is usually referred to as weak [11]. Finding such a context-dependent plan is a much simpler task than computing a strong contingent plan with conditional branches which would satisfy the goal in all possible state sequences that could arise from observational effects. Postponing the computation of alternative contingent branches until more information is acquired from the environment is a more feasible approach for Web service scenarios that involve many numeric variables and output-to-input mappings, and being optimistic is likely to pay off. Thus, all sources of non-determinism, where the actual behaviour of actions at execution time contradicts the expected effects as modelled in the planning domain or external agents alter the world in unanticipated ways, are left to be treated by interleaving planning with execution and performing continual planning.

¹ www.w3.org/TR/wsdl.

3.1 Representing the Planning Domain and Encoding it as a Constraint Satisfaction Problem

To deal with incomplete knowledge and sensing, the planning domain description is enriched with a knowledge-level representation to model observational actions (sensing effects). Conditional effects are also provided. Moreover, the planning formalism accommodates for numeric functions and effects beyond mere assignments, such as increase/decrease. Another characteristic of the planning schema is that input arguments to actions may range over numeric-valued domains just as all other variables. Since the planning problem is translated into a CSP, the supported variable domains (integer, real, lower and upper bounds) depend on the underlying constraint solver that is used. The planning domain accommodated by the RuGPlanner is described briefly in Definition 1 (for the complete consistent definition see [19]).

Definition 1 (Planning Domain). *A Planning Domain is a tuple $\mathcal{PD} = \langle \text{Var}, \text{Par}, \text{Act} \rangle$, where:*

- *Var is a set of variables ranging over a finite domain.*
- *Par is a set of variables that play the role of input parameters to members of Act ranging over a finite domain.*
- *A is the set of actions. An action $a \in \text{Act}$ is a quadruple $a = (\text{id}(a), \text{in}(a), \text{precond}(a), \text{effects}(a))$, where:*
 - *$\text{id}(a)$ is a unique identifier*
 - *$\text{in}(a) \subset \text{Par}$ are the input parameters of a*
 - *$\text{precond}(a)$ is a propositional formula over $\text{Var} \cup \text{Par}$*
 - *$\text{effects}(a)$ is a conjunction of effects. An effect may refer to assigning to a variable some value or some other variable; increasing or decreasing a variable by some constant; sensing a variable, i.e. observing its current (unknown) value; an “invalidate” effect that states that a variable becomes unknown; or a conditional effect that is applied at the next state only if a proposition holds at the current state.*

Following a common practice in many planning approaches, we consider a *bounded* planning problem, i.e., we restrict our target to finding a plan of length at most k for an a priori given integer k . In the following, we explain how the service domain is encoded into a CSP, for some given integer k . The process is similar to the one described in [16] (alternative encodings based on the planning graph are proposed in [13]). Considering a planning domain $\mathcal{PD}$, the aim is to encode it into a $\text{CSP} = \langle X, \mathcal{D}, \mathcal{C} \rangle$, where X is a finite set of variables, $\mathcal{D}$ is the set of finite domains of the variables in X , and $\mathcal{C}$ a finite set of constraints,

i.e. propositions that restrict the allowable values of variables in X . First, the variables X are derived as follows: for each variable $x \in V \cup Par$ ranging over D^x , and for each $0 \leq i \leq k$, we define a CSP variable $x[i]$ in CSP with domain D^x . Actions are also represented as variables: for each action $a \in Act$ and for each $0 \leq i \leq k-1$ a boolean variable $a[i]$ is defined. This way the computed plan can include parallel actions, a fact that may save time at execution. After deriving the CSP state variables X , the actions' preconditions and effects are also encoded into constraints as described in [19]. On top of the domain description, restrictions referring to the initial state are expressed in the form of a conjunction of propositions. A strong requirement that all variables involved in the preconditions should be known is also added as part of the precondition constraints, to ensure that the preconditions hold for all possible assignments to variables consistent with their allowed domain at a given state. Frame axiom constraints are also generated, which guarantee that variables cannot change between subsequent states, unless some action that affects them takes place.

The set of constraints that comprise CSP are further extended by additional constraints that constitute the goal (see Sect. 3.2), yielding the planning problem in the form of a CSP that is passed to a standard constraint solver. It should be noted that due to the handling of the sensing effects, which allows the offline solver to assume the existence of “convenient” values for unknown variables, the planner generates an optimistic plan. This means that the offline plan anticipates that all knowledge-gathering actions return information that is in accordance with the user's requirements, and all actions are executed successfully. During execution this optimistic initial plan may be revised, as described in Sect. 3.3.

3.2 Goal Language

Until now, we have described the representation and encoding into a CSP of the planning domain and the initial state. In the following, we present the syntax and semantics of the goal, and show how this can be translated into a set of constraints which together with the constraints formulating the planning domain and initial state constitute the final CSP which is passed to the constraint solver.

The goal language supported by the RuGPlanner equips the user with potent constructs for expressing complex goals, beyond the mere statement of properties that should hold in the final state. Conditions over state traversals, maintainability properties, and distinguishing between the wish to observe the environment and the wish to change it are some of the features this language supports. These aspects are expressed in a declarative way, so that the user does not have to know about the operational details of the available operations and how they can be combined. The basic goal operators have been first presented in [17], while many constructs of its formalisation resemble the syntax presented in [23].

Goal Syntax. The goal syntax is defined as follows:

$$\begin{aligned}
 \text{goal} & ::= \bigwedge_i (\text{condition-goal}_i \mid \text{condition_or_not-goal}_i \\
 & \quad \mid \text{subgoal}_i) \\
 \text{condition-goal} & ::= (\text{subgoal}) \text{ under_condition goal} \\
 \text{condition_or_not-goal} & ::= (\text{subgoal}) \text{ under_condition_or_not goal} \\
 \text{subgoal} & ::= \text{final}(\text{props}) \mid \\
 & \quad \text{achieve}(\text{props}) \mid \\
 & \quad \text{achieve-maint}(\text{props}) \mid \\
 & \quad \text{all_states}(\text{props}) \mid \\
 & \quad \text{find_out-maint}(\text{props}) \mid \\
 & \quad \text{find_out}(\text{props})
 \end{aligned}$$

where *props* is a propositional formula as the *precond*(*a*) defined in Definition 1, with $\text{var}, \text{var}_1, \dots, \text{var}_n \in (\text{Var} \cup \text{Par})$. All variables and parameters not specified in the goal (or the initial state constraints) are assumed to be undefined (i.e., their respective knowledge-level variables are set to false).

The **final** subgoal is satisfied if *props* holds at the last state, while **achieve** requires that *props* should be true at *some* state over the state traversal. The **maint** annotation adds the requirement that once the respective propositions become true at some state, they should remain true in all subsequent states until the final one. **all_states** imposes that *props* should be true at all states, and is usually applied on input parameters whose values are set by the user. The **find_out** type of subgoals enforces a hands-off requirement on the variables the respective propositions involve, i.e., the planner tries to satisfy the propositions at some state without allowing any world-altering effect on these variables before that state. **find_out-maint** ensures that the involved variables should remain intact at all states of the plan. For instance, the goal **find_out**(*account_balance* > 100) will be satisfied if the sensed value of the variable *account_balance* is greater than 100, without allowing any action to alter the value of the variable before the sensing action. On the other hand, if the goal is **achieve**(*account_balance* > 100), the planner will do everything possible in order to fulfil the proposition, e.g., it might invoke a *pay_in* action that increases the *account_balance* by some amount.

Subgoals can be further on combined through the condition goal constructs, which impose some conditions that should be assured before the fulfilment of the subsequent subgoal. *subgoal*₀ **under_condition** *goal*₁ is satisfied if *subgoal*₀ is satisfied for the first time at some state *s* and *goal*₁ is satisfied in the state sequence preceding *s*. **under_condition** thus imposes a before-then relation between goals over the state traversal, and is particularly useful in cases where the user would like to go ahead with altering some variable, only if its sensed value satisfies some property beforehand. **under_condition_or_not** allows the expression of what can be seen as some kind of soft requirements: *subgoal*₀ **under_condition_or_not** *goal*₁ will also be fulfilled if *goal*₁ is not satisfiable; if it is, however, then *subgoal*₀ has to be as well. It should be mentioned that the **under_condition_or_not** structure works as intended only if the variables involved in *goal*₁ are known at planning time. The formal semantics of the goal language is provided in [19].

Example Goals. The following presents three simple example goals to demonstrate the use of some basic constructs supported by the goal language:

Goal 1

```
achieve-maint(bookedConcert = TRUE) under_condition
  (find_out-maint(temperature > 0))
```

Goal 2

```
achieve-maint(bookedHotel = TRUE) ∧ (
achieve-maint(bookedConcert = TRUE)
  under_condition_or_not (find_out-maint(temperature > 0)))
```

Goal 3

```
 $\bigwedge_i$  achieve(robotLocation = roomi)
```

Goal 1 is accomplished if s is the first state at which $bookedConcert = TRUE$ is satisfied, and $find_out_maint(temperature > 0)$ is satisfied in the state sequence preceding s (in this example, the maintainability requirement imposed by SPSVERBc15SPSVERBc3 is in practice redundant because there is no way to change the weather). If $temperature < 0$, then Goal 1 fails. On the other hand, Goal 2 ensures that $bookedConcert = TRUE$ will be satisfied if the temperature is not below zero, while if it is, then only $bookedHotel = TRUE$ will be looked after. Goal 3 expresses that a robot should visit *all* rooms in a house, leaving the order of visits to be computed by the planner depending on the structure of the house.

3.3 Orchestration: Interweaving Plan Synthesis with Execution

The suitability of the RuGPlanner for dynamic service environments lies in the idea of delaying the computation of alternative plans until these plans are called for by the new information acquired during execution. Continual planning is performed, so that the upcoming plan steps anticipated offline can be revised as execution progresses, in face of inconsistencies that arise either from the newly acquired information, from services' inconsistent behaviours or from the actions of exogenous agents that interfere with the plan. In such a setting, the goal is considered to have been accomplished, if all individual actions in this sequence of updated plans are successfully executed. We call the process that starts from an initial plan and moves on by interweaving action invocations with plan revisions an *orchestration*. Orchestration is performed by applying gradual modifications to the CSP instance which models the planning domain, the goal and the constantly changing contextual state. The modifications correspond to the incorporation of new facts about the state of the world or the removal of obsolete ones, to refinements of the sequence of actions included in the already computed plan, or to useful information about the behaviour of services that is collected by inspecting how they operate. The orchestration algorithm is characterised by the following traits:

- It exploits the high degree of parallelism in the plan, by performing concurrent invocations and handling the responses in a non-blocking way. Since the execution time of some service operations may be very long, the orchestrator is able to continue planning and proceed to the execution of subsequent actions if this is allowed by the domain and goal restrictions, while waiting for the response of a service invocation.
- It provides the means to recover from failure responses and timeouts. Other arbitrary service outcomes that contradict its expected behaviour can also be tolerated under the assumption of a consistent and timely publish-subscribe mechanism.
- It can cope with possible discrepancies due to the activity of exogenous agents, which may act in parallel with the plan execution and interfere with it.
- It seeks to keep a balance between the effort spent in computing a new plan from scratch and in refining an existing one.
- It takes care of the data flow by instantiating numeric-valued input parameters to the actually sensed output parameters. This is performed through plan refinement, by considering the history of known facts in the form of constraints and the goal requirements.

More information about the orchestration algorithm used by the RuGPlanner can be found in [19].

4 Application: Automated Service Composition in a Smart Home

Pervasive computing environments such as our future homes are the prototypical example of a dynamic, complex system where Service-Oriented Computing techniques will play an important role. Ubiquitous computing technologies can be used to assist people to accomplish their desired tasks, increase their safety and feeling of comfort, and enhance the level of independence of people with functional disabilities by enabling them to access and control their household devices. This vision of homes which exhibit a high degree of intelligence brings a number of fresh challenges, including support for interoperation, dynamic discovery, sensing of the current execution context, and dynamic coordination of the several smart artefacts. A Service-Oriented Architecture, where smart components are exposed as services, can contribute towards addressing these challenges, and fits naturally in dynamic environments, where the location, connectivity, and availability of a large number of autonomous and heterogeneous objects constantly change during the home's life cycle.

To satisfy the wishes of the user and guarantee their comfort and safety, the house has to be able to exhibit quite complex functionalities rather than just triggering some single service or a predesigned sequence of fixed services. A trivial operation such as turning on a light in a corridor can be achieved with a switch or a passive infrared sensor. However, the coordination of the home so as to effectively deal with a gas leak detection is far more demanding, especially

when considering the many possible contextual states the house and the user can be in, each of which may require several possible handlings to achieve the same ultimate safety goal. Moreover, developing rigid solutions that are tailored to a specific home instance and user needs is not an efficient approach, given the considerable effort that is required to adapt them for new customers.

In order for smart homes to exhibit a genuinely intelligent behaviour, the ability to compute compositions of individual devices automatically and dynamically is paramount. In this context, the application of appropriate AI planning techniques can address challenges put forward by dynamic domestic settings, offering a high degree of customisability to different home instances and user needs and being able to perform complex reasoning about contextual information rather than relying on hardwired sets of service instances. The application of the RuGPlanner in smart home environments [9, 20, 21, 32] has been particularly promising, being able to effectively address challenges related to dealing with numeric values, extended goals, and dynamicity considering an evolving set of elements typical of the Internet of Things.

4.1 A Motivating Example

Let us consider a home inhabitant who is a disabled person who can move around on an electric wheel-chair, while a nurse pays a visit for some hours every day. A location component keeps track of the location of the users to the level of some predefined areas. Let us see how a home equipped with a service composition architecture that makes use of the RuGPlanner would behave.

At 8 pm the waking-up goal prescribed by the user is automatically triggered: the alarm clock rings, the curtains in the bedroom are opened, the lights may be turned on depending on the amount of daylight detected by a natural light sensor, and the motorised bed is elevated. After taking a shower, the user wants to move to the sitting room and watch some TV. Such a goal dictates that the TV is set to the user's channel of preference, the lights are adjusted depending on the indication of the natural light sensor, and the curtains are also shut accordingly. The air-conditioner is turned on if the temperature sensor in the living room indicates that the temperature is too high, while the necessary doors are opened to facilitate the user moving to the sitting room. At noon, the user goes to the kitchen to prepare something to eat. While being there, the smoke detector in the kitchen identifies a potentially dangerous smoke leak—but fortunately not due to fire. As a result, a predefined home goal for dealing with this situation is automatically triggered: after having ensured that the user has safely moved out of the kitchen (let's say to the adjacent sitting room), the door leading to the kitchen is closed to isolate the smoke in a single room. The ventilator is turned on and the kitchen window is also opened, so that the foul air is expelled, while an alarm notification appears on the TV screen. While waiting in the sitting room, the user wants to move back to the kitchen, but only after having been assured that the environment there is safe, and the smoke has been eliminated. This wish implies resorting to sensing to identify the current situation in the

kitchen. Let's assume that after some time the smoke is eliminated, causing the alarm on the TV and the ventilator to automatically turn off.

Later in the afternoon, while the user is taking a bath, and the nurse has gone out for some shopping, a fall is identified by the fall detector attached to him, and an emergency goal is automatically triggered: the health centre is notified and an informative message is sent to the nurse's mobile phone, while the robot is moved to the bathroom in case the user wants to ask for some additional assistance. Given that the fall has not caused any serious trouble, the nurse finds the user lying in his bed reading a book, and after some time he decides that it's time for going to sleep. He thus issues a goal that prepares the bedroom conditions for sleeping, which involves setting the alarm clock to some preferred wake-up time, lowering the motorised bed position, turning off the lights, and closing the curtains.

The aforementioned scenarios assume that no contingencies occur during execution, and that all service invocations complete successfully. What if, however, a service is out of order, and responds with a failure or if a timeout occurs? In such cases, the system will first try to reinvoke the erroneous service, and if again a failure or timeout is observed, it will perform replanning. Let us assume, for example, that, when executing the plan that prepares the living room for watching TV, the automatic turning on operation of the TV service responds with a failure. Assuming that the robot assistant is also endowed with the capability of turning on the TV by manually pressing the button on the device, the composition engine will compute an alternative plan which involves moving the robot in front of the TV so that it can switch it on.

5 Service Composition for Large-Scale Domains

Smart environments such as smart buildings and, at a larger scale, smart cities combine a set of services, primarily consisting of sensors and actuators, to enable automatic control of various processes within the environment with the goal of maintaining a certain desired state of the environment while respecting the preferences of individual users. The highly dynamic nature of these types of problems requires powerful reasoning capabilities, making planning, represented as a constraint satisfaction problem, particularly suitable for modelling them. A representation of the environment consists of uncontrollable variables, whose values cannot be controlled directly but can only be observed through the continuous stream of data provided by the sensors, and variables whose state can be controlled directly, which correspond to the actuators. The values of the controllable variables are determined based on the current state of the environment and the rules governing the environment, thereby performing certain actions within the environment, which may influence the state of the uncontrollable variables, to reach or maintain the desired state. The order in which actions are performed is, unlike many traditional planning problems, often of less importance, because most of the actions do not directly depend on other actions but rather on the state of the environment.

Fig. 1. The deployment of a small-scale smart building system in the Bernoulliborg of the University of Groningen.

Building automation systems are a typical example of smart environments. Sensors installed in the building provide information about the current state of the environment, including the light level and temperature in different parts of the building and occupancy information. Using this information, the system controls processes within the building such as lighting, heating and ventilation as well as individual workstations and appliances. The aim of the system is to provide the occupants of the building with a comfortable place to work by satisfying the preferences of the occupants, such as the desired temperature in a room, while, for example, minimising the energy consumption of the building. Energy savings can be realised by switching off appliances when they are no longer in use or lighting when no occupancy is detected. Integration with a smart grid may additionally allow the building to reduce the overall operational cost by buying energy at times that the prices are lower. The GreenMind project, which presents an architecture and case study of a smart building system, exemplifies the realisation of a small-scale smart building system deployed across several offices [25]. The deployment, shown in Fig. 1, consists of various sensors and actuators to control the power consumption of workstations and other appliances based on the presence of occupants to minimise the energy consumption of the building. As demonstrated by this project, such smart building systems can save a substantial amount of energy, even at a small scale.

The exponential complexity of constraints satisfaction problems makes creating planning problems for large-scale domains such as these difficult because it is not possible to find a solution to the problem in a reasonable amount of time. A reduction of the complexity can be achieved by transforming the constraints into a different representation in which a function of some sensors implies a state of the actuators that must be satisfied [12]. This representation uses the fact that the values of variables corresponding to sensors cannot be changed directly, and introduces a notion of activeness for the constraints. Constraints that are

inactive for a particular state of the environment can be removed from the problem, reducing both the computational complexity and, more importantly, the dependencies between different parts of the problem.

Some large-scale domains also have an inherently localised structure. For a smart building, for example, the rooms of the building may serve as a natural boundary for many of the constraints, causing the preferences of any particular occupant to be mostly confined to their office or workspace within the building. As a result, these constraints form small clusters, or islands, of more densely connected constraints, with only some constraints crossing the boundary of such an island. This structure can be made an explicit part of the problem by reformulating the problem as a CSP consisting of sparsely interconnected islands of constraints, where islands correspond to the parts of the problem that are more densely connected and bridges represent the sparse dependencies between them.

Definition 2. Let $\mathcal{P} = \langle \mathcal{I}, \mathcal{B} \rangle$ be a CSP consisting of a set of islands $\mathcal{I}$ and a set of bridges $\mathcal{B}$. The set of islands $\mathcal{I} = \{I_1, \dots, I_n\}$ consists of independent CSPs $I_i \in \mathcal{I} = (X_i, D_i, C_i)$. Islands are completely independent, meaning that $I_i, I_j \in \mathcal{I}, i \neq j : X_i \cap X_j = \emptyset \wedge C_i \cap C_j = \emptyset$. The bridges $\mathcal{B} \subseteq \{B_{ij} \mid i, j \in \{1, \dots, n\} \wedge i \neq j\}$ connect islands together, where a single bridge $B_{ij} \in \mathcal{B} = (S_{B_{ij}}, r_{B_{ij}})$ is a constraint between two islands $I_i, I_j \in \mathcal{I}$ with $S_{B_{ij}} \subseteq X_i \cup X_j$.

For a problem of this type, inactive constraints have the potential to partition the problem into several independent subproblems: when some of the constraints corresponding to the bridges become inactive, the dependencies between the islands are removed, resulting in one or more independent subproblems. For each of these problems, a solution can be constructed independently, which can ultimately be combined into a single solution to the complete problem. Previous solutions can also be reused when the state of the environment changes. If a part of the problem remains independent despite the changes in the environment, its solution remains unchanged, making it unnecessary to repeat the search process. The parts of the problem that are affected by the changes in the environment may have to be adjusted, but, as for the complete problem, the previous solution can be used as a starting point for the search, potentially reducing the search time considerably.

6 Conclusion

In this chapter we looked at one of the research directions which originated from a joint work of one of the authors of the paper and Mike Papazoglou. The goal of this chapter was to show how a vision of Mike and the idea of automated service composition has developed over the years, how it has been adjusted and applied to different domains, and how it is still actively developed.

In this work, we first looked into the problem of automated service composition, specifically a composition of services according to a rich (extended) goal language. Traditional planning approaches assume a simple propositional formula representing a set of goal states. With the extended goals you can define

more complex properties of the desired plan: order of actions, properties that need to be maintained in all visited states, etc. In practical applications you may not know all the information in advance, for example, the customer may define the shipment address only after he selected and confirmed the products he plans to buy. To model such incomplete information, a new method of interleaving planning and execution was developed, with uncertainty being explicitly modelled via knowledge-gathering actions.

When applied to a domain of Internet of Things, and specifically to smart homes and smart offices, additional assumptions have been made: instead of uniform services, the domain is described via explicit sensors and actuators explicitly representing the state of the domain and active world-altering actions. Unfortunately, solving planning problems for such domains is very difficult due to the computational complexity of the planning algorithms and typical sizes of the domains. Luckily, the structure of the domains allows for a reduction of the complexity, as many world state changes do not necessarily influence the whole domain: often local state changes would imply local planning and actuation. Constraints that are inactive for a particular state of the environment can then be removed from the problem, reducing both the computational complexity and, more importantly, the dependencies between different parts of the problem.

References

1. Aiello, M.: *The Web Was Done by Amateurs: A Reflection on One of the Largest Collective Systems Ever Engineered*, 1st edn. Springer, Cham (2018). <https://doi.org/10.1007/978-3-319-90008-7>
2. Aiello, M., et al.: A request language for web-services based on planning and constraint satisfaction. In: Buchmann, A., Fiege, L., Casati, F., Hsu, M.-C., Shan, M.-C. (eds.) *TES 2002*. LNCS, vol. 2444, pp. 76–85. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46121-3_10
3. Au, T.-C., Kuter, U., Nau, D.: Web service composition with volatile information. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) *ISWC 2005*. LNCS, vol. 3729, pp. 52–66. Springer, Heidelberg (2005). https://doi.org/10.1007/11574620_7
4. van Beest, N., Kaldeli, E., Bulanov, P., Wortmann, J., Lazovik, A.: Automatic detection of business process interference. In: *1st International Workshop on Knowledge-Intensive Business Processes (KIBP)*, 13th International Conference on Principles of Knowledge Representation and Reasoning (KR) (2012)
5. van Beest, N., Kaldeli, E., Bulanov, P., Wortmann, J., Lazovik, A.: Automated runtime repair of business processes. *Inf. Syst.* **39**, 45–79 (2014)
6. Berardi, D., Cheikh, F., De Giacomo, G., Patrizi, F.: Automatic service composition via simulation. *Int. J. Found. Comput. Sci.* **19**(2), 429–451 (2008)
7. Bertoli, P., Kazhamiakin, R., Paolucci, M., Pistore, M., Raik, H., Wagner, M.: Continuous orchestration of web services via planning. In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI (2009)
8. Bertoli, P., Pistore, M., Traverso, P.: Automated composition of web services via planning in asynchronous domains. *Artif. Intell.* **174**, 316–361 (2010)

9. Caruso, M., Ciccio, C.D., Iacomussi, E., Kaldeli, E., Lazovik, A., Mecella, M.: Service ecologies for home/building automation. In: Proceedings of the 10th IFAC Symposium on Robot Control (2012)
10. Casati, F., Sayal, M., Shan, M.-C.: Developing E-services for composing E-services. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 171–186. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45341-5_12
11. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.* **147**(1–2), 35–84 (2003)
12. Degeler, V., Lazovik, A.: Dynamic constraint satisfaction with space reduction in smart environments. *Int. J. Artif. Intell. Tools* **23**(06), 1460027 (2014). <https://doi.org/10.1142/S0218213014600276>
13. Do, M.B., Kambhampati, S.: Planning as constraint satisfaction: solving the planning-graph by compiling it into CSP. *Artif. Intell.* **132**, 151–182 (2001)
14. Georgievski, I., Nguyen, T.A., Aiello, M.: Combining activity recognition and AI planning for energy-saving offices. In: 2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing, pp. 238–245 (2013)
15. Ghallab, M., Laruelle, H.: Representation and control in IxTeT, a temporal planner. In: Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems, pp. 61–67 (1994)
16. Ghallab, M., Nau, D., Traverso, P.: *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam (2004)
17. Kaldeli, E., Lazovik, A., Aiello, M.: Extended goals for composing services. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009). AAAI Press (2009)
18. Kaldeli, E., Lazovik, A., Aiello, M.: Continual planning with sensing for web service composition. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence. AAAI Press (2011)
19. Kaldeli, E., Lazovik, A., Aiello, M.: Domain-independent planning for services in uncertain and dynamic environments. *Artif. Intell.* **236**, 30–64 (2016)
20. Kaldeli, E., Warriach, E., Lazovik, A., Aiello, M.: Coordinating the Web of services for a smart home. *ACM Trans. Web* **7**(2), 10 (2013)
21. Kaldeli, E., Warriach, E.U., Bresser, J., Lazovik, A., Aiello, M.: Interoperation, composition and simulation of services at home. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 167–181. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17358-5_12
22. Klusch, M., Renner, K.U.: Fast dynamic re-planning of composite OWL-S services. In: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT) - Workshops, pp. 134–137 (2006)
23. Lazovik, A., Aiello, M., Gennari, R.: Encoding requests to web service compositions as constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 782–786. Springer, Heidelberg (2005). https://doi.org/10.1007/11564751_64
24. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 629–643. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68234-9_46
25. Nizamic, F., Nguyen, T.A., Lazovik, A., Aiello, M.: GreenMind - an architecture and realization for energy smart buildings. In: Proceedings of the 2014 Conference ICT for Sustainability, pp. 20–29. Atlantis Press, August 2014. <https://doi.org/10.2991/ict4s-14.2014.3>

26. Ouvans, C., Dumas, M., ter Hofstede, A., van der Aalst, W.: From BPMN process models to BPEL web services. In: International Conference on Web Services, pp. 285–292 (2006)
27. Papazoglou, M.P., Aiello, M., Pistore, M., Yang, J.: Planning for requests against web services. *IEEE Data Eng. Bull.* **25**(4), 41–46 (2002). <http://sites.computer.org/debull/A02DEC-CD.pdf>
28. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B.J.: Service-oriented computing: a research roadmap. In: Curbera, F., Krämer, B.J., Papazoglou, M.P. (eds.) *Service Oriented Computing (SOC)*. Dagstuhl Seminar Proceedings, vol. 05462, 15–18 November 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2005). <http://drops.dagstuhl.de/opus/volltexte/2006/524>
29. Peer, J.: A POP-based replanning agent for automatic web service composition. In: Gómez-Pérez, A., Euzenat, J. (eds.) *ESWC 2005*. LNCS, vol. 3532, pp. 47–61. Springer, Heidelberg (2005). https://doi.org/10.1007/11431053_4
30. Sheshagiri, M., DesJardins, M., Finin, T.: A planner for composing services described in DAML-S. In: *Proceedings of the 13th ICAPS Workshop on Planning for Web Services* (2003)
31. Sohrabi, S., Baier, J.A., McIlraith, S.A.: Preferred explanations: theory and generation via planning. In: *Proceedings of the 25th AAAI Conference on Artificial Intelligence* (2011)
32. Warriach, E.U., Kaldeli, E., Bresser, J., Lazovik, A., Aiello, M.: A tool for integrating pervasive services and simulating their composition. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) *ICSOC 2010*. LNCS, vol. 6470, pp. 726–727. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17358-5_74