Tay *et al. Robomech J (2016) 3:17*
DOI 10.1186/s40648-016-0056-0

ROBOMECH Journal

RESEARCH ARTICLE

Open Access

Service robot planning via solving constraint satisfaction problem

Noel Nuo Wi Tay*, Azhar Aulia Saputra, János Botzheim and Naoyuki Kubota

Abstract

The problem of demographic shifts towards the elderly is deteriorating, as the relative number of caregivers is insufficient to provide the support required for their wellbeing, which is further aggravated by the increasingly hectic lifestyle. Service robot is getting more prominent as a possible solution. Robot manipulation and mobility is an important field, but they also require high level planning for these minute actions in order to provide ample support. Automatic service composition, contributed significantly by web services, offers the necessary technology for the task. Robot planning problem can be solved by representing it as constraint satisfaction problem (CSP) due to it being able to support loose binding of services and variables of wider domain. This paper further extends the structure of the CSP planner to enable intelligent decision-making. Besides, standardization is made on the properties and relationships of objects such that planning rules can be easily generated from knowledge base. Services and their corresponding terms are designed for efficient planning. Case studies show that the system is able to perform tasks bounded by complex logical rules. It also provides valuable insights into future enhancements and research.

Keywords: Planning, CSP, Robot

Background

The fraction of the elderly on the population pie is increasing at an alarming level for developed countries. It is estimated that the population of those above 60 years old in the world will double to 2 billion in the year 2050 [1]. The problem of demographic shifts towards the elderly is particularly serious in Japan [2, 3], as the relative number of caregivers is insufficient to provide the support required for their wellbeing. The issue is further aggravated by the increasingly hectic lifestyle.

Smart homes and ad-hoc home automations are on the rise to provide support for the elderly. Despite that, installation and integration of the system requires technical knowledge. Besides, large variety of vendors and standards brings more confusion instead of peace to the elderly. We by no means say that smart home and home automation fail to deliver. In fact, they are necessary in the near future to provide the required assistance. At the current stage, it is more appropriate to devise a physical agent that can dispense service at a home yet to be supported or supported minimally by smart devices, and, at the same time, allowing the evolution of the home to take place to become a more autonomous environment. This agent, or we shall call, service robot, should provide security, assistance, communication and companionship to its human occupants.

Various service robots have been developed over the years. An example is the Human Service Robot (HSR) developed by Toyota [4]. Robot manipulation and mobility is an important field for service robot. These are low level actions, where they require high level activity planning in order to perform complex tasks. Service planning is able to provide them with the high level overview of how their minute actions will be carried out overtime.

Different approaches are used for service planning. Service composition is initially applied to create composite services for various complex business requirements. Services can be considered the building blocks of what an agent can undertake at a certain time. To fulfill a goal, the agent should execute a sequence of services. Overtime, the idea is adopted by ubiquitous, pervasive computing and robotic. In order to support service composition,

*Correspondence: tay-noelnuowi@ed.tmu.ac.jp
Graduate School of System Design, Tokyo Metropolitan University, Hino, Tokyo, Japan

services come with semantic markups that describe their functionalities, properties and conditions (pre and post conditions of the service). With the semantic markup, it is much more convenient to plan, reason and monitor services, instead of relying on detailed descriptions of the functions. It endows them with capabilities of being a planning operator.

In the field of robotics, Markov decision process (MDP) is a popular approach for robot motion planning. The downside of this approach is that it lacks the complexity required for service composition, which can be delivered by logic-based approach [5]. The fact that MDP is a statistical method gives them an edge in this respect in dealing with uncertainty over logic-based approach. Therefore, both statistical and logical approach can be considered tackling different scope of the same problem. In this paper, we will concentrate on the service composition performed by automated reasoning. It handles the broad service composition such that it also provides room for statistical or other specialized methods to figure out the minute details.

In [6], OWLS-Xplan uses the semantic descriptions of services defined in OWL-S for planning purposes. The XPlan planning module will then generate the composite services. In this work, an XML dialect of planning domain definition language (PDDL) is developed. This makes the system PDDL compliant. Although the system obtains semantic descriptions in OWL-S, it is not utilized and semantic awareness is not achieved. In this case, the planning module is required to perform exact matching for service inputs and outputs.

In [7, 8] a framework was developed that converts service composition tasks into planning problems expressed in PDDL. The framework will then convert the devised plan into an OWL-S composite process description. The framework translates atomic OWL-S processes to planning operators. Goals are achieved through assembling these generated planning operators. When no exact composite services can be found, semantic information is utilized to obtain composite services that best approximate the goal.

To deal with complex tasks and to reduce planning complexity, Hierarchical Task Network (HTN) [9] is introduced. It uses method definitions in its planning domain description, which specifies how the complex tasks can be broken down into more manageable tasks [10, 11]. The planning problem can then be specified as a list of tasks to perform. The planner will then solve the problem by applying the breaking down of tasks to every task in the task list. This process continues until the tasks are reduced to their atomic planning operator constituents that corresponds to a solution plan. The advantage is its speed. In spite of that, the disadvantage is that the planning process requires certain decomposition rules be specified due to its hierarchical nature. This means that it needs to be encoded in advance by an expert.

Answer set programming (ASP) is another popular approach used for planning [12, 13]. It is a declarative language that is suitable for knowledge representation and non-monotonic reasoning. [12] integrated ASP with cost learning to improve the performance of the planner for robot planning, while [13] combines with MDP to endow it with the capability to handle uncertainty. At the moment, for complex sequence of services, these methods require heavy computational load and long planning time.

The methods discussed thus far require exact match ups between inputs, outputs and variables, or that it assumes certain ontologies to handle heterogeneities, or requires specifications of user anticipation and procedural templates. Domain and goal modeling through constraint satisfaction problem (CSP) is developed to create a language that allows users to express goals without having to know about the details and interdependencies between services [14–16]. Its domain representation is of similar concept with the multi-valued planning task (MPT) encoding [17]. Besides, another advantage is that it is able to handle variables with large domain efficiently, which is quite prevalent in the field of autonomous home such as temperature value and user location. Although the CSP planner might be slower than some state of the art methods [11, 18, 19], it can support complex goals and can handle variables with large domain efficiently. It has been applied to robot planning [15]. But the CSP planner proposed does not show intelligence in making choices as argued in [20]. Besides, no standardization is made to function/predicate construction and domain specification, which prevents automatic generation of rules.

In this paper, robot planning problem is represented as CSP, where sequence of plans is generated for the robot to execute to fulfill goals. Through this approach, robot individual services become loosely coupled, thus, enabling more flexibility and enhancing reusability in service design. Flexibility is achieved in the sense that the design of services does not need to take into account the details and functionalities of other services - they only need to know the preconditions before and effects of the states after service execution, where it is up to the CSP planner to arrange them to obtain desirable plans to fulfill goals. Being reusable means the services can be applied to different applications without having to manually redefine rules pertaining planning, as in the redefinition of subgoals from HTN methods. Besides, constraint programming supports variable of larger domain more efficiently, which allows natural processing of constraints involving

values such as integers and large finite domain data types. In this case, goals and rules involving numbers (for example, air conditioner need to be switched on if temperature rise above certain value) can be combined with robot activity planner under one umbrella of declarative language.

This paper further extends the CSP planning structure proposed in [15] to support making choices of different objects as part of planning process, by endowing dynamic response variables in functions and predicates with finite domain data types. Besides, standardization is made in determining sub-locations for robot mobility as well as function and predicate specifications relevant to objects of the house, where all these can be easily generated automatically from knowledge base inference (for example, generating relevant objects and their classes to be filled into planning functions and predicates from SPARQL query or making inference through OWL description logic from home knowledge base. Interested readers can refer to ontology building from [21]). Original work in [15] does not provide such specifications (such as how sub-locations should be defined) and grounds the functions and predicates (such as the grounded terms for beer and its property). One may argue that all functions and predicates can ultimately be grounded, but this comes at a price of (1) understandability and (2) CSP planning optimization. Grounded terms are harder to interpret, at least during development stage, and thus, compromise understandability, where information transfer from knowledge base is less intuitive. Grounded terms may impose additional constraints on CSP based planner and additional difficulty for optimization. As CSP solver can be built to specialize in planning tasks, it does not need grounded terms as in Boolean Satisfiability and may even exploit functional forms for faster heuristic search as well as supporting weighted CSP, which is subject of future work. This paper provides a much more complicated home environment case, yet at the same time, employing standardized approach in service design that can be easily generated automatically and applied to other environments.

Methods

Planning and execution of plans (termed orchestrator) are performed by a single mobile robot. The plans involve high level plans like where the robot should go to, opening doors, picking up certain objects and so on. It does not involve minute details such as how the robot should open doors through the coordination of servos and visual input. In this work, when a robot is expected to execute a plan, the robot is already in an environment that is conducive or possible to execute such plans. The planner and the orchestrator will execute a sequence of plans to prepare such an environment. For example, when the robot needs to switch on a light, the planner will first tell the robot to go near the corresponding switch before switching it on with its manipulators.

For this work, a few basic movements of the robot are assumed, which are, mobility, tuning up/down, opening and closing, switching on and off, picking and putting objects. It is assumed that the robot can hold something while performing open/close and tuning up/down actions. Planning and execution will revolve around these basic movements to achieve certain goals. Planner module is responsible for the planning and passing commands to the robot for its execution. Given a certain goal, planner module will compose a sequence of plans for the robot to execute such that after the plans are completed, the goal is achieved. For example, given that the goal is to place a canned soft drink in the fridge and that all cabinet needs to be closed, the robot will first find where the soft drink is. If there is one that happens to be in the kitchen cabinet, the robot will approach it, open the cabinet and pick up the soft drink, and then close it. It will subsequently approach and open the fridge, after which it will place the canned drink before closing it. No prior programming is required for the plan.

This work emphasizes on generating plans, thus, time to achieve this is assumed to be acceptable. It is also assumed that the syntactic statement used by the planner for the planning problem describes the actual environment. Therefore, failure or success of robot in achieving its task is reflected in the environment of planning problem.

Planner module

Figure 1 shows the flow chart of the planning and execution process. Three important components of the planner module are the variables, services and goals. Variables record the current state of robot and the environment. Services are activities that the robot can perform, which have preconditions and effects. Precondition contain a list of conditions that need to be fulfilled by the variables before the plan can be executed, whereas effect is the changes that will occur after the service is implemented. The goal is the final condition that needs to be met by the variables. Planner module consists of a planner and an orchestrator. Planner will compose a sequence of plans according to current variable state. If a solution is found by the planner, the plan will be passed on to the orchestrator to execute the plan. Orchestrator's role is to pass commands to the robot to execute the plans accordingly.

Given the available variables, services and goals, the planner will compose a sequence of plans such that, after implementation, the goal will be fulfilled. The sequence of plans can be likened to theorem proving [22]. Every

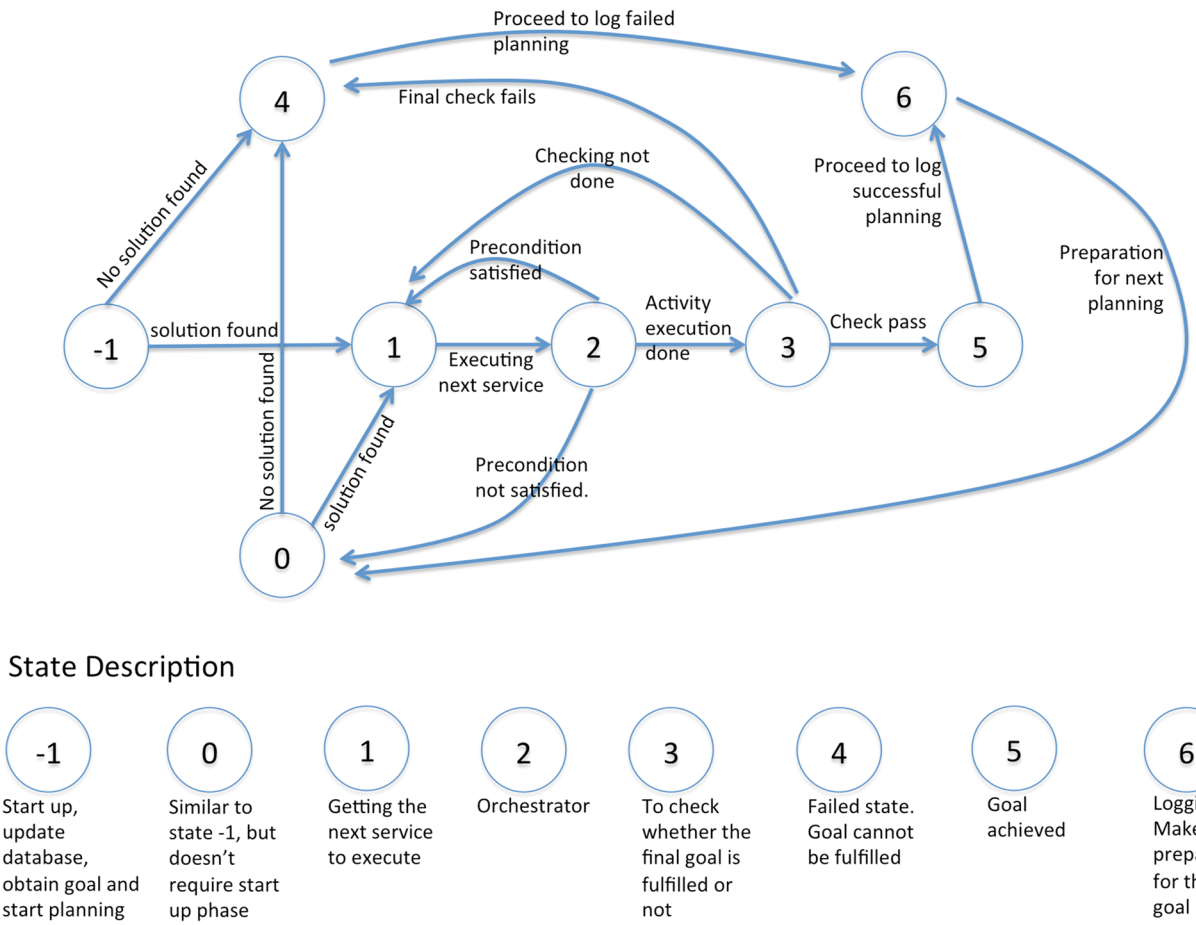

Fig. 1 Planning and execution flow. When the program starts, the state is *-1*. Given the goal, the state will change to *1* if solution is found, *4* otherwise. Execution is done through state *1* to state *3*. When the plan execution is finished, the state will proceed to state *5* and *6*, upon which new goal will be obtained for subsequent planning

subsequent plans will have their preconditions met, and will impose changes for the next plan to drive the variables to what is required by the goal.

The following is an explanation of the flow chart in Fig. 1. When the program starts, the state will be at state -1 (note that the use of the word 'State' here is only for explanation in Fig. 1. State -1 only occurs once when the program starts. During this state, start-up booting will occur as well as extracting crucial information from the knowledge base. The first goal is also obtained at this stage. Planning will occur to devise a sequence of plans to fulfill the goal. If a solution is found, the program will proceed to state 1. Otherwise, it will go to state 4, signifying a failure in finding solution. The purpose of state 1 is to obtain the next service in the plan for execution, after which will be executed at state 2. State 2 implements as the orchestrator. It first checks whether preconditions of the current activity are met or not. If not, the program will proceed to state 0, or else, the service will be executed and returns to state 1 to pick up the next service. State 0 resembles state -1, just that it doesn't require start up phase. The transition from state 2 to state 0 enables dynamic planning. This is important to deal with uncertain situation, where variables cannot be confirmed except during run-time. In this case, re-planning can occur at state 0 if expectation is wrong. After plan execution, state 3 will re-check whether the goal is fulfilled, upon which is yes, it will proceed to state 5. If the goal cannot be achieved, the program will proceed to state 4. The final state is state 6 which records a log for future reference or learning. The program will restart at state 0 after selecting the next goal.

General domain description

Domain description will be based on the work of [15]. We denote ϑ = term set (list of variables). ϑ contains V terms, which consists of knowledge, effect, dynamic response and static response terms confined by their

own domain. Term can be a variable, constant or function. Response terms represent information that can only be obtained from objects, information that comes from sources not within ϑ. During planning stage, static response terms remain the same throughout all planning sequence and represent an unknown value (thus, initialization constraint is imposed on them), whereas dynamic response can change for every sequence. They can take on whatever value to facilitate constraint satisfaction during planning that employs an optimistic approach (value taken on by response terms are considered true). Dynamic response terms can also be determined or having their domain constrained by current, past or future variables via the use of rules (In this work, only past and current variables are considered). Response term usage will be made more evident in subsequent sections. Knowledge terms record information for future reference. Effect terms are used for external control.

A state is a tuple of values to terms at a particular plan implementation sequence index t that is denoted as $X_t = (X_t^1, X_t^2 ... X_t^V)$ where $X_t^1, X_t^2 ... X_t^V \in \vartheta_t$, confined by their domains denoted by $D^1, D^2 ... D^V$. As there is a finite limit to the number of sequence per plan being planned denoted as K, thus, $0 \leq t < K$. For the current work, domains of the variables of the terms remain unchanged over time.

α is the set of activities, where $a = (id(a), precond(a), effect(a)) \in \alpha$. $id(a)$ is the identifier of the activity. There is an additional activity in α that does nothing. It has no pre-conditions and effects, termed as *Nop*.

$precond(a)$ is the pre-condition that need to be met before the activity can be executed, such as the robot needs to near the cabinet before attempting to open it. Precondition of an activity can be described as follows:

$$\begin{aligned} precond(a) ::= prop | precond(a) \wedge precond(a) | precond(a) \\ \vee precond(a) | \neg precond(a) | precond(a) \\ \rightarrow precond(a) \end{aligned} \tag{1}$$

$$\begin{aligned} prop ::= var \bullet var | var \bullet val | (var \odot var) \\ \bullet var | (var \odot var) \bullet val | nPred \end{aligned} \tag{2}$$

where $var \in \vartheta$, val is a constant, $\odot \in \{+, -\}$ is a binary operator, $\bullet \in \{=, <, >, \neq, \leq, \geq\}$ is a relational operator, and *nPred* is an n-ary predicate.

$effect(a)$ is the changes that will be induced after the activity is completed. It emulates the state transition given the activity a, such that its logical formulation can be used to impose constraints on subsequent sequence of the plan for activity planning. It should be emphasized that the actual object manipulates variables during run-time after planning instead of the $effect(a)$ formulation (which is only used for planning). Effect of an activity can be formulated as or a combination of the following: $var_{t+1} = val$, $var_{t+1} = var_t$, $var_{t+1} = f(v_1, v_2)$ where $v_1, v_2 \in \vartheta_t$ or v_1, v_2 are constants, and f is the sum, subtraction and Boolean operation.

For simplification, when necessary, we will denote the above relations as $var_{t+1} = effects_t(a)$. This relation is read differently between the planner and the orchestrator. During planning, $var_{t+1} = effects_t(a)$ means the truth statement that: ($effects_t(a)$ includes an effect towards the variable var) implies ($var_{t+1} = effects_t(a)$) holds true.

On the contrary, for orchestrator, it is seen as $var_{t+1} := effects_t(a)$, which indicates that the term var is being modified according to $effects_t(a)$. Therefore, depending on whether planner or orchestrator is referred to, the correct interpretation has to be made. There is no arrow of time for planner, thus, the relation is seen as equality. For orchestrator, it is seen as an assignment.

Though this work only use the specified effects, more sophisticated effects, such as conditional effects, can be used as shown in [15]. That said, the extension to previous work [15] is shown in Fig. 2, indicated by the red bounding box. The extension is the use of dynamic response terms and the rules for their transition. This extension enables inference capabilities and the ability to make choices to be realized as part of planning process. As dynamic response terms and their corresponding rules are dependent on the activity the robot is performing, detailed explanation will be given in "Design of services" section.

Planning as constraint satisfaction problem

Given goals, which are represented as propositions, activity planning can be obtained to fulfill the goal by representing the problem as CSP and solve it. A CSP is a triple $CSP = \langle \chi, D, \zeta \rangle$, where χ is a set of terms, D is the set of domains of the variables of the terms in χ, and ζ is a set of constraints over χ. A solution to a CSP is an assignment of values to the terms in χ such that the values fall within D and all constraints in ζ are satisfied. In this work, D is unchanged throughout the activity sequence. It is considered determined when a goal is passed to the planning process flow in Fig. 1, and will stay that way until the goal is achieved.

In terms of the ISS planner, $\chi = \{X_1, X_2 ... X_K\} \cup \{A_1, A_2 ... A_{K-1}\} \cup R$, R is a set of response terms, and A_t is the chosen goal at sequence index t. Unlike X and A, variables in R remain the same throughout the planning sequence.

ζ consists of constraints imposed by a chosen activity at t from activity pre-conditions and effects, inertia law, initial and final variable state and maintenance of achieved goal constraints. Initial variable state is just a constraint

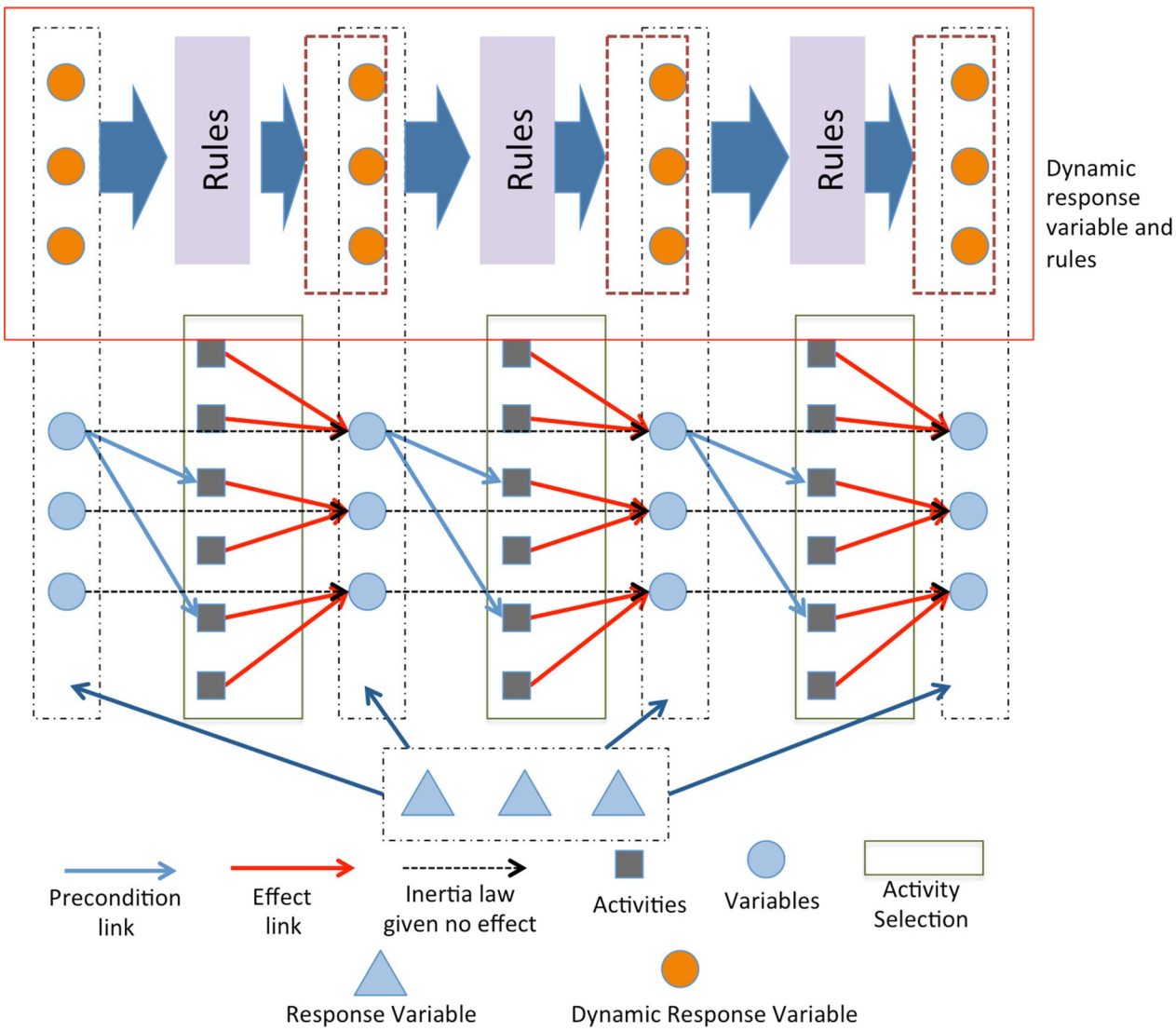

Fig. 2 Constraint graph used for robot planning. Although there is no explicit definition of time in CSP planning, the *arrows* show the flow of plan. The *red bounding box* shows the extension of this work from that of [15], which consists of dynamic response variable and rules that dictates the values of subsequent dynamic response variables

that dictates the values of all variables (obtained from object state module) before any planning. Final state constraint consists of the goal proposition that needs to hold at sequence index K.

Constraints from activity pre-conditions:

$(A_t = a) \rightarrow precond(a)$ where $\forall a \in \alpha$

Constraints from activity effects:

$(A_t = a) \rightarrow [(var_{t+1} = effects_t(a)) \wedge Fr]$ where $\forall a \in \alpha$ where Fr is the inertia law constraint, which indicates that for every other variables var (excluding those from R) not affected by $effects_t(a)$, $var_{t+1} = var_t$.

Maintenance of achieved goal constraint:

This constraint dictates that whenever a goal is achieved at sequence index $\bar{t} < K$:

$A_t = Nop$ where $\bar{t} < t < (K-1)$

Maintenance of achieved goal constraint is just one of the goals specified in [15], though it is sufficient for the current work.

Constraints are fed to a solver to obtain a sequence of A, which are the activities that need to be implemented to fulfill the given goals. Z3, which is a state of the art SMT solver, is used to obtain the plan [23]. The plan will be solved by continually increasing K until the constraints are satisfied.

Design of terms

The robot in this work is intended to fetch and place objects, as well as opening/closing and switching on/off switches. It needs to be able to move around to enable it to perform the tasks. There are three types of objects, which are, movable objects, non-movable objects and location.

Movable objects consist of objects that can be moved around by the robot like towel and cans. Non-movable objects are objects that are fixed, but they may be able to be operated by the robot. Examples of non-movable objects are bed, doors, cabinet and switches. The robot itself is also considered a non-movable object for convenience during planning which will be shown later on. Human is also considered a non-movable object as the assumption is that the human remain stationary during planning. If the human moves, due to the ability to re-plan as discussed previously, the issue can be easily solved. Location object consists of regions on the home such as the living room, bedroom and kitchen.

Apart from integer and boolean datatype, five new data types are introduced to support the mentioned objects, namely, *NOType*, *MOType*, *Location*, *NOStateType*, *MOStateType*, *NOID* and *MOID*. *NOType* defines the type (ex: cabinet, door, switch) of non-movable object with ID defined in data type *NOID*. Likewise, *MOType* defines the type (towel, paper, cup) of movable object with ID defined in data type *MOID*. *Location* is the datatype of location.*MOStateType* and *NOStateType* are the datatype that defines the state of a movable and non-movable object respectively.

There are two static functions (functions where the output values remain the same throughout all planning sequence), which are, $FNOType : NOID \rightarrow NOType$ and $FMOType : MOID \rightarrow MOType$. A constant is *HumanLocation*, and a static predicate is *NOLocation*(*NOID*, *Location*).

FNOType maps a particular non-movable object to its type (For example, mapping an object as a door). Like wise, the same applies to *FMOType*, but that it applies to movable object. The predicate *NOLocation*(a, b) states the truth value whether a non-movable object a is in location b. This is especially important for objects like doors. *HumanLocation* stores where the human is at in datatype *Location*

The subsequent terms explained in this subsection can have their values changed in the course of planning sequence. This means, given a term A, for a plan with K sequence, there will be $A \times K$ number of variable A, each for every sequence, where each has a unique definition $A_1, A_2 ... A_K$.

The function $DYStateNO_t : NOID \rightarrow NOStateType$ outputs the state of the non-movable object. $DYStateNO_t(a) = b$ states that a is in a state of b at sequence t. The same applies to $DYStateNO_t : MOID \rightarrow MOStateType$, which is for movable objects.

The function $DYTune_t : NOID \rightarrow Int$ outputs the numerical value (in integer *Int* type) associated with the non-movable object at sequence t.

As movable objects will always be placed at a non-movable object (ex: cup on a table, book with a human), the function $DYAtMO_t : MOID \rightarrow NOID$ maps a movable object to a non-movable object it is placed at.

Three variables are included, namely, $RobotLocation_t$, $RobotApproach_t$ and $RobotHold_t$. $RobotLocation_t$ stores the location of the robot (with datatype *Location*) at sequence t. $RobotApproach_t$ records the non-movable object the robot is approaching. $RobotHold_t$ is a boolean variable determining whether the robot is holding something or not.

For proper functionality, three dynamic response variables are introduced, which are, $Approachres_t$, $MOIDres_t$ and $RobotHoldres_t$. The role of these dynamic response variables will be made more evident during the discussion of the services.

Design of services

This subsection describes, but not limited to, three types of services the robot is expected to perform. Although only three types are listed, more services can be included depending on the application. The four types are open/closing service, tuning up/down service, mobility, and put/pick service.

Open/closing service consists of two services, which are opening and closing. These two services applies to opening/closing of doors and switching on/off switches.

It has a precondition:

$NOLocation(RobotApproach_t, RobotLocation_t) = true$

and effect:

$DYStateNO_{t+1}(RobotApproach_t) := Open/Close$ respectively

The precondition has to make sure the robot is approaching object stored in $RobotApproach_t$ and that the robot is in location *RobotLocation*, before it can open/close the non-movable object by manipulating *DYStateNO* at sequence $t + 1$.

Tuning up/down service deals with numerical manipulation that consists of (1) tuning up (increasing by 1), and (2) tuning down (decreasing by 1).

It has a precondition:

$NOLocation(RobotApproach_t, RobotLocation_t) = true$

and effect:

$DYTune_{t+1}(RobotApproach_t) := DYTune_t(RobotApproach_t) + / - 1$

respectively

Put/Pick service consists of robot fetching and placing a movable object.

The preconditions for picking up objects are:
$DYStateNO_t(RobotApproach_t) = Open$
$DYAtMO_t(MOIDres_t) = RobotApproach_t$
$NOLocation(RobotApproach_t, RobotLocation_t) = true$
$RobotHold_t = false$
and effects:
$DYAtMO_{t+1}(MOIDres_t) := NRobot$
$RobotHold_{t+1} := true$

The preconditions make sure that the movable object the robot is trying to fetch is located in a non-movable object with state *Open* through the predicate $DYStateNO_t(RobotApproach_t)$,and that the robot is not holding anything via $RobotHold_t = false$. They also make sure the object the robot is trying to fetch ($MOIDres_t$) is located in non-movable object $RobotApproach_t$. $MOIDres_t$ is a dynamic response variable, where its value is freely determined by the planner to aid optimistic planning which is inherent in planning via solving CSP. If the preconditions are met, the robot can pick object $MOIDres_t$ through setting $DYAtMO_{t+1}(MOIDres_t)$ to *NRobot*, where *NRobot* is the non-movable object ID under datatype *NOID* for the robot. *RobotHold* is set to true to indicate that the robot is holding something.

The preconditions for putting objects are:
$DYStateNO_t(RobotApproach_t) = Open$
$DYAtMO_t(RobotHoldres_t) = NRobot$
$NOLocation(RobotApproach_t, RobotLocation_t) = true$
$RobotHold_t = true$
and effects:
$DYAtMO_{t+1}(RobotHoldres_t) := RobotApproach_t$
$RobotHold_{t+1} := false$

For putting objects, the preconditions also specify that the intended non-movable object be opened. It requires the robot to hold movable object $RobotHoldres_t$, which is indicated by having *NRobot* as the output for function $DYAtMO_t$. Just like $MOIDres_t$, $RobotHoldres_t$ is a dynamic response variable that stores the current object the robot is holding. $RobotHold_t = true$ is the constraint where the robot is holding something. With the preconditions met, $RobotHoldres_t$ will be placed at $RobotApproach_t$ through $DYAtMO_{t+1}$.

Mobility service consists of two services, that is, movement within a room, and movement between rooms.

Movement within a room has the following precondition:
$NOLocation(Approachres_t, RobotLocation_t) = true$
and effect:
$RobotApproach_{t+1} := Approachres_t$

The robot will always be going towards a non-movable object, as it is practically meaningless to go towards nothing. Therefore, the precondition makes sure that a non-movable object the robot is going to is within the current room, where the dynamic response variable $Approachres_t$ stores the object the robot is approaching.

Movement between rooms is required for every doors. One can write a precondition as the following:

$$\exists z(NOLocation(z, RobotLocation_t) \wedge NOLocation(z, res_t) \wedge (res_t \neq RobotLocation_t) \wedge DYStateNO_t(RobotApproach_t) \wedge (RobotApproach_t = z) \wedge (FNOType(z) = Door))$$

This precondition with existential quantifier takes a longer time to plan from preliminary test. Another alternative is to build a service for every doors. The precondition is shown as follows:
$DYStateNO_t(V1) = true$
$RobotLocation_t = V2$
and effects:
$RobotLocation_{t+1} := V2$

Services with above mentioned preconditions and effects are built for every door and for every side. It means that if there are two doors altogether, there are a total of four such services, where each door takes up two for the robot to move through the door in both directions. From the preconditions and effects, $V1$ is the door object ID and $V2$ is the location the robot goes to after passing the door.

Results and discussion

This section presents the case study of the planner for robot service execution. The intention is to show the applicability of the approach, while at the same time, provides insights into future developments.

Case study will show how the robot, given the standard activities (mobility, open/close and pick/put), can fulfill goals that can provide support for its human inhabitants. It is done through simulation built via open dynamics engine (ODE). Orchestrator from the planning module constantly communicates with the robot. It sends service instructions one at a time for the robot to execute, while the robot will respond given every executed service. The study is run on 2.5 GHz Intel Core i5 computer.

House setup

Case study is performed on a simulated home via ODE. Fig. 3 shows the layout of the house. It contains 36 non-movable objects (excluding human and the robot), 7 movable objects, 10 locations, and 10 doors. The numbers in white shown in the figure are the non-movable object IDs.

Out of the all the non-movable object, the bed (ID 1) and table (ID 34) cannot be manipulated. Doors, windows, fridge, cabinets and washing machine can be

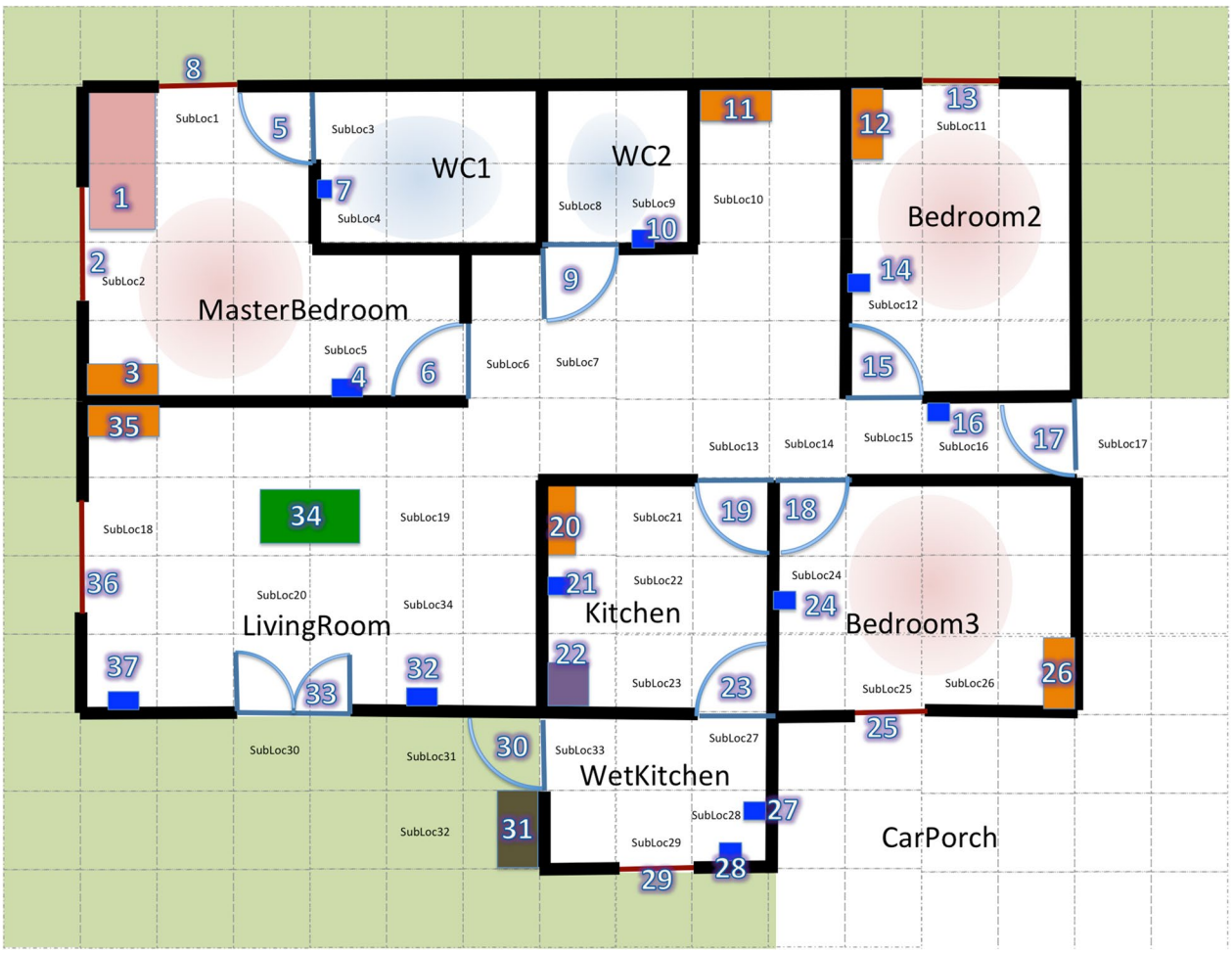

Fig. 3 House layout. The house layout used for robot planning simulation. *Blue parts* are switches, *red lines* are windows, *orange parts* are cabinets, *Purple part* is a fridge, green part is the table, *pink part* is the bed and *brown part* is the washing machine. The *shapes with swinging curves* are doors

opened/closed. Apart from doors, all non-movable objects only have one associated location (indicated by predicate *NOLocation*). Of all the non-movable objects, only N37 is associated with *DYTune*, as N37 is considered a volume control for the living room fan. The locations are, Master bedroom, Living room, Kitchen, Wet Kitchen, WC1, WC2, Bed room 2, Bed room 3, Car Porch and the Garden.

Numbers preceded by 'subloc' are sublocations for the objects. Since the robot need to be near enough to an object before it can manage it, the sublocations provide such spots. Sublocations are not included in the planning, but their information with their corresponding non-movable objects are stored in the knowledge base such that they can be used during execution. This information is crucial during path planning. For this work, due to the simple grid-like layout, Floyd-Warshall algorithm is used for path planning.

Details of the movable objects are shown in Table 1, which shows their type and which non-movable objects they are placed at. Information for these movable objects are also extracted from knowledge base of the home. For faster planning, one can limit the information to only deal with the objects required.

For clarity, all IDs for non-movable object will be preceded by a capitalized N, and all movable object will be preceded by M during the case studies. For example, the door with ID 6 will be identified as N6, while the towel with ID 5 in the washing machine N31 will be identified by M5.

Speed comparison on different service design

Service design will have significant effect on how search is performed to obtain solution. In this section, test is performed to select whether to utilize a general service that covers wide number of objects, or to duplicate the

Table 1 Movable object details

Object ID	MOType	NOID
1	Canned drink	22
2	Canned drink	20
3	Canned drink	12
4	Paper	34
5	Towel	3
6	Towel	31
7	Book	11

services to handle these objects individually. Although speed is not the main focus of this paper, but optimization is required to make it general enough to cover wide range of applications without specialized tweaking yet fast enough for most environments.

That said, we will concentrate on two types of services, that is, moving between locations (S1) and opening/closing (S2) services. Both of these service types covers a wide range of objects, thus is important to select the optimal approach to implement them to obtain the best speed. We denote type A test as general service for S1 and S2, type B test as general S1 and duplicated S2, and finally type C test as duplicated S1 and general S2. For clarity, explicit command means the goal is explicitly specified (ex: moving object A to B), while implicit command means the goal is implied (ex: move any object with property N out of D). Type A contains 6 services, type B contains 25 services and type C has 91 services. Table 2 shows the test result on the speed of planning for all three types of services.

It can be observed that type B (general S1 and duplicated S2) achieves the best result most of the time. Type A is the slowest most of the time despite it having only 6 services to choose and search from, which shows generality comes at a price on speed. Type C, having the most services at 91, beats type A in planning speed. Despite that, its performance is insignificant compared to type B, due to its high number of duplicated services. Therefore, the subsequent case studies will utilize type B as design approach for the services. Table 3 shows the details of all type B services.

Summary of cases

This section briefly describes the six case studies used to demonstrate the capabilities of the planning system.

Case 1: "Simple object fetch for human" is used as a starting example on fulfilling an explicit goal of robot fetching a book for its human master. It is also used to show a difference between implicit and explicit goal.
Case 2: "Dynamic planning under uncertain situation" shows how the planner deals with uncertain situation that causes inconsistencies. Uncertain situation may cause information that is crucial for planning to change without a prior update on the planner's knowledge. This case demonstrates dynamic re-planning to tackle such issues.
Case 3: "Inferences for making choices" demonstrates more clearly (compared to Case 1) the use of implicit goals for the planner to make choices based on the properties of objects, which is also an extended capability from previous work [15].
Case 4: "Reasoning and planning with numbers" demonstrates how the CSP planner treats numerical reasoning and manipulation as part of the planning process.
Case 5: "Reusability of activities" shows how the robot activities can easily co-operate with the activity of other smart devices installed in the smart home. It reuses the activity definition of the smart device without having to make further manipulations.
Case 6: "Complex goals" shows planning under more complex implicit goals and its implications in time and enhancements.

Case 1: Simple object fetch for human

A service robot need to be able to approach human as well as fetching objects or them or picking or putting objects according to command, without the user having to dealt into too much detail of how it is done. This

Table 2 Test on Different Service Design

Test no.	Description of the goal	Command	Steps	A (sec)	B (sec)	C (sec)
1	Approach N1	Explicit	4	0.431	0.219	0.294
2	Open all windows in master bedroom	Implicit	7	1.357	0.938	1.346
3	Move M2 to N20 (same location)	Explicit	9	2.27	1.03	1.76
4	Move any canned drink to M34	Implicit	9	3.36	2.74	5.21
5	Move M5 to N31 (different location)	Explicit	13	18.36	14.94	11.36
6	Switch on N10, N12, N14, N28	Explicit	22	655.40	272.67	609.35

Table 3 Service preconditions and effects

No.	Precondition	Effect
1	$NOLocation(RobotApproach_t, RobotLocation_t)$	$DYStateNO_{t+1}(RobotApproach_t) := true$
2	$NOLocation(RobotApproach_t, RobotLocation_t)$	$DYStateNO_{t+1}(RobotApproach_t) := false$
3	$NOLocation(Approachres_t, RobotLocation_t)$	$RobotApproach_{t+1} := Approachres_t$
4	$DYStateNO_t(N6) \wedge RobotLocation_t = MasterBedroom$	$RobotLocation_{t+1} := LivingRoom$
5	$DYStateNO_t(N6) \wedge RobotLocation_t = LivingRoom$	$RobotLocation_{t+1} := MasterBedroom$
6	$DYStateNO_t(N5) \wedge RobotLocation_t = MasterBedroom$	$RobotLocation_{t+1} := WC1$
7	$DYStateNO_t(N6) \wedge RobotLocation_t = WC1$	$RobotLocation_{t+1} := MasterBedroom$
8	$DYStateNO_t(N9) \wedge RobotLocation_t = WC2$	$RobotLocation_{t+1} := LivingRoom$
9	$DYStateNO_t(N9) \wedge RobotLocation_t = LivingRoom$	$RobotLocation_{t+1} := WC2$
10	$DYStateNO_t(N15) \wedge RobotLocation_t = Bedroom2$	$RobotLocation_{t+1} := LivingRoom$
11	$DYStateNO_t(N15) \wedge RobotLocation_t = LivingRoom$	$RobotLocation_{t+1} := Bedroom2$
12	$DYStateNO_t(N17) \wedge RobotLocation_t = LivingRoom$	$RobotLocation_{t+1} := CarPorch$
13	$DYStateNO_t(N17) \wedge RobotLocation_t = CarPorch$	$RobotLocation_{t+1} := LivingRoom$
14	$DYStateNO_t(N18) \wedge RobotLocation_t = LivingRoom$	$RobotLocation_{t+1} := Bedroom3$
15	$DYStateNO_t(N18) \wedge RobotLocation_t = Bedroom3$	$RobotLocation_{t+1} := LivingRoom$
16	$DYStateNO_t(N19) \wedge RobotLocation_t = LivingRoom$	$RobotLocation_{t+1} := Kitchen$
17	$DYStateNO_t(N19) \wedge RobotLocation_t = Kitchen$	$RobotLocation_{t+1} := LivingRoom$
18	$DYStateNO_t(N23) \wedge RobotLocation_t = Kitchen$	$RobotLocation_{t+1} := WetKitchen$
19	$DYStateNO_t(N23) \wedge RobotLocation_t = WetKitchen$	$RobotLocation_{t+1} := Kitchen$
20	$DYStateNO_t(N30) \wedge RobotLocation_t = WetKitchen$	$RobotLocation_{t+1} := Garden$
21	$DYStateNO_t(N30) \wedge RobotLocation_t = Garden$	$RobotLocation_{t+1} := WetKitchen$
22	$DYStateNO_t(N33) \wedge RobotLocation_t = LivingRoom$	$RobotLocation_{t+1} := Garden$
23	$DYStateNO_t(N33) \wedge RobotLocation_t = Garden$	$RobotLocation_{t+1} := LivingRoom$
24	$DYStateNO_t(RobotApproach_t) \wedge$ $DYAtMO_t(MOIDres_t) = RobotApproach_t \wedge$ $NOLocation(RobotApproach_t, RobotLocation_t) \wedge$ $\neg RobotHold_t$	$DYAtMO_{t+1}(MOIDres_t) := NRobot \wedge$ $RobotHold_{t+1} := true$
25	$DYStateNO_t(RobotApproach_t) \wedge$ $DYAtMO_t(RobotHoldres_t) = NRobot \wedge$ $NOLocation(RobotApproach_t, RobotLocation_t) \wedge$ $RobotHold_t$	$DYAtMO_{t+1}(RobotHoldres_t) := RobotApproach_t \wedge$ $RobotHold_{t+1} := false$
26	$NOLocation(RobotApproach_t, RobotLocation_t)$	$DYTune_{t+1}(RobotApproach_t) := DYTune_t(RobotApproach_t) + 1$
27	$NOLocation(RobotApproach_t, RobotLocation_t)$	$DYTune_{t+1}(RobotApproach_t) := DYTune_t(RobotApproach_t) - 1$

case study on object fetch showcases how the robot is able to find the book (from knowledge of the knowledge base) and deliver it to the user, who is in the bedroom. This case covers the aforementioned requirements of the robot.

Case 1 assumes simple goals:

$DYAtMO_K(M7) = NHuman$

$\forall z((FNOType(z) = Cabinet) \rightarrow \neg DYStateNO_k(z))$

where the first goal states that the book with ID M7 will be given to the human, and the second goal requires all cabinet to be closed at the end of the plan. The robot initial location is at non-movable object N17.

The following shows the planning:

Case 1

Approach N6⇒ Open N6⇒ Approach N11⇒ Open N11⇒ Pick up M7 from N11⇒ Close N11⇒ Pass Door N6 to MasterBedroom⇒ Approach NHuman⇒ Place M7

Planning time= 1.335 seconds

Figure 4 shows the robot's plan execution. Given the generated plan, the robot will first approach the door N6 to open it. It will then go to cabinet N11 to pick up the book M7. After that, the robot move to the bedroom to deliver the book to the human. A peculiarity that occurs is the fact that the robot opens door N6 first, and then returns to Cabinet N11, before going back to N6 to enter the Master bedroom. This is because the planner doesn't know about the cost of paths. It only judge the

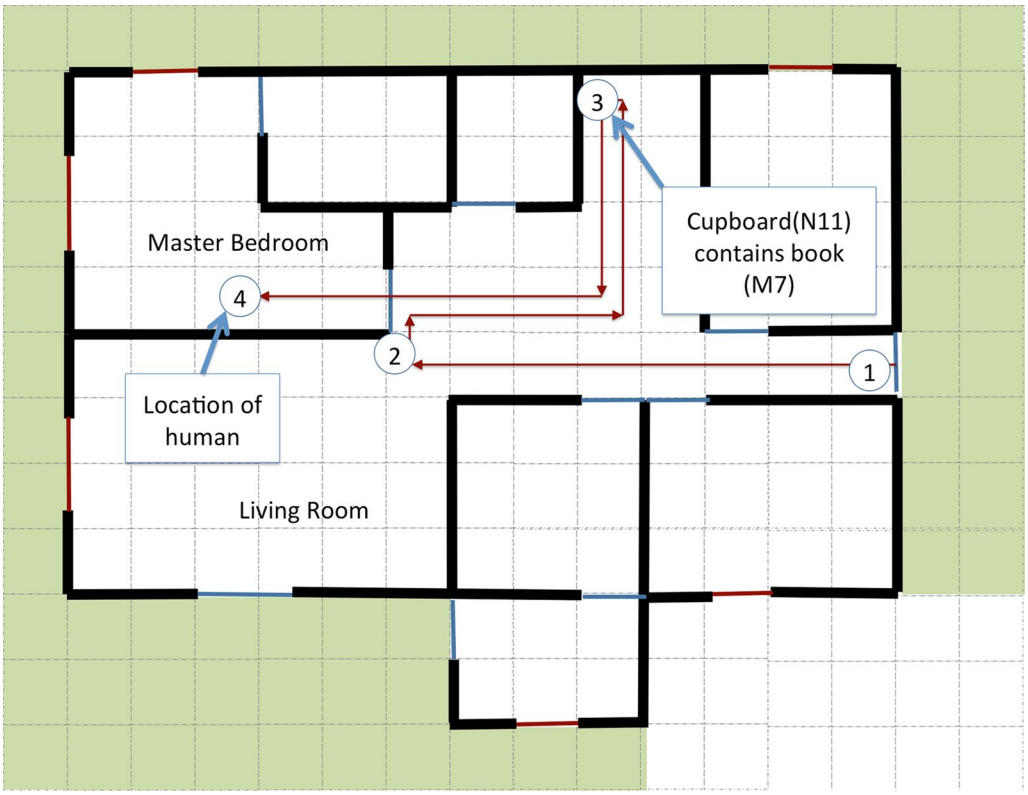

Fig. 4 Case 1: Simple object fetch for human. Step 1 is the starting point. The robot proceeds to master bedroom door(N6) and opened it in Step 2. It then goes to the cupboard (N11), opens it, and gets a book, and closed the cupboard in Step 3. Finally, in Step 4, it goes into the master bedroom and passes the book to the human

shortest number of activities based on the basic services aforementioned.

An implicit goal of fetching the book can also be carried out. Case 1 explicitly states which book to deliver to the human. One can also set up an implicit goal as follows:

$$\exists z((FMOType(z) = Book) \wedge (DYAtMO_K(z) = NHuman))$$

where the planner will find any book to be delivered to the human. Sequence of plans remains the same, except that the average time of planning is 1.941 seconds, as it covers a wider search space.

Case 2: Dynamic planning under uncertain situation

Planning needs to take into account the fact that the environment will change in the course of planning or plan execution. The planner explained in "Planner module" section supports dynamic re-planning, which means, if inconsistency is met, re-planning can take place, taking into account current information.

In this case study, the robot needs to fetch the human (who is in the living room) a canned drink. At the start of the experiment, as shown in Table 1, 3 canned drinks are present in the home. During the experiment, we simulate the condition where a person takes away the canned drink that the robot is after just before it reaches for it, except for the last (or 3rd) canned drink. Case 2 goal is simple as follows:

$$\exists z((FMOType(z) = CanDrink) \wedge (DYAtMO_K(z) = NHuman))$$

The generated plan (and re-planning) is shown as follows:

Case 2

Approach N19⇒ Open N19⇒ Pass Door N19 to the Kitchen⇒ Approach N22⇒

Open N22⇒ Pick up M1 from N22⇒ Pass Door N19 to the Living Room⇒ Approach

NHuman⇒ Place M1 at NHuman

Planning time = 1.82 s

Inconsistency during execution at "Pick up M1 from N22". Re-planning is performed...

Approach N20⇒ Open N20⇒ Pick up M2 from N20⇒ Pass Door N19 to the Living Room⇒ Approach NHuman⇒ Place M2 at NHuman

Planning time = 1.14 s

Inconsistency during execution at "Pick up M2 from N20". Re-planning is performed.

Pass Door N19 to the Living Room⇒ Approach N15⇒ Open N15⇒ Pass Door N15 to Bedroom 2⇒ Approach N12⇒ Open N12⇒ Pick up M3 from N12⇒ Pass Door N15 to the Living Room⇒ Approach NHuman⇒ Place M3 at NHuman

Planning time= 2.17 s

Figure 5 shows the robot execution to fulfill the goal. At the initial stage, there are three canned drinks, two in the kitchen (with MOID M1 and M2) and one in bedroom two (with MOID M3). The robot's initial plan is to fetch M1 from the fridge in the kitchen. Right before it can fetch the drink, someone takes it. While trying to pick an object, a precondition for the activity is $DYAtMO_t(MOIDres_t) = RobotApproach_t$ that requires the object the robot wants to fetch to be in the location the robot approaches. Therefore, a missing M1 means the precondition cannot be fulfilled, and thus, from Fig. 1, this will lead to re-planning. Re-planning occurs until the robot manages to get a canned drink and passes it to the human.

Case 3: Inferences for making choices

As an example, when human gives command to turn on the light in the living room, considering there are multiple lights, he/she would convey something like "Turn on a dim light in the living room", instead of "Turn on light A32". In this example, it doesn't matter which light is being turned on, as long as it is a dim light. Therefore, properties of objects are much more crucial. This case demonstrates the capability of the planner to make intelligent choices based on the properties of objects it needs to handle, instead of explicit definition of objects.

The goal is to fetch an object of type "Towel" and located in the garden, and transfer it to cabinet N11. In our case study, our home has two towels M5 (in the master bedroom) and M6 (in the garden). In this goal, no explicit definition of the object ID is specified. Instead the planner needs to select these objects based on the stated properties. The goals are shown as follows:

$$\exists x, y((FMOType(x) = Towel) \wedge (DYAtMO_0(x) = y) \wedge NOLocation(y, Garden) \wedge (DYAtMO_K(x) = N11))$$

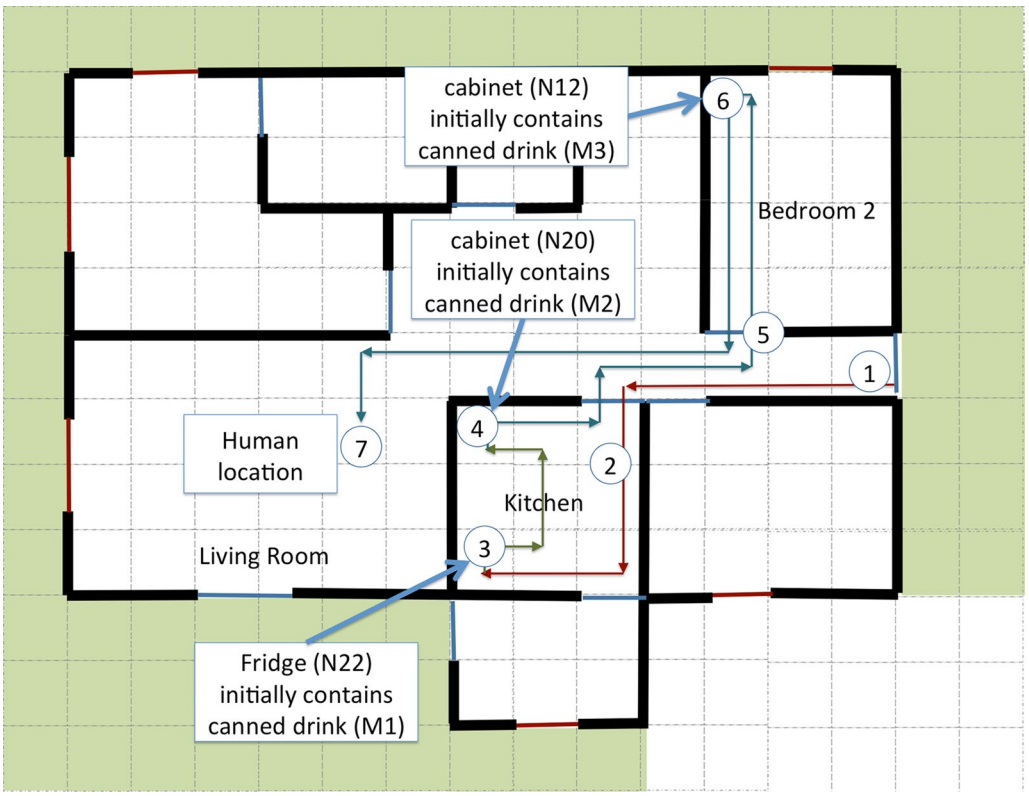

Fig. 5 Case 2: Dynamic planning under uncertain situation. Different colors of path trajectories shows different plans, where *red* represents executed the first plan, which subsequently leads to *green* (due to missing M1), after which leads to the *blue* (due to missing M2). From step 1 to 3, the robot tries to fetch M1 from the fridge N22. In step 3, due to missing M1 (due to someone taking it away), re-planning is performed, which leads the robot to fetch M2 from the cabinet(N20) in step 4. As M2 is missing too, re-planning is performed, which leads to the robot fetching M3. It goes to bedroom 2 in Step 5, and fetch M3 in the cabinet in Step 6. Finally, it brings it to the human in the living room in Step 7

The generated plan is as follows:

Case 3

Approach N33⇒ Open N33⇒ Pass Door N33 to the Garden⇒ Approach N31⇒ Open N31⇒ Pick up M6 from N31⇒ Pass Door N33 to the Living Room⇒ Approach N11⇒ Open N11⇒ Place M6 at N11

Planning time= 1.36 s

Executing the plan, the robot will transfer towel M6 in the washing machine to N11. To get towel from the master bedroom, all one needs to do is to change the location to MasterBedroom in the goal as follows:

$$\exists x, y((FMOType(x) = Towel) \wedge (DYAtMO_0(x) = y) \wedge NOLocation(y, MasterBedroom) \wedge (DYAtMO_K(x) = N11))$$

This demonstrates the ability to make intelligent choices, where making inferences becomes part of the planning process, which is an extension from [15] (where choices need to be explicitly made). ASP approach [12, 13] can provide such descriptive power, but currently it is still new in the field of planning, and it cannot directly handle numbers as shown in Case 4.

Case 4: Reasoning and planning with numbers

In alot of cases, manipulation and reasoning with numbers are required, such as the case in volume control and triggers from numerical constraints. CSP planner is able to cover these domains under one declarative language. This case study will demonstrate this capability through tuning the fan volume based on the temperature. Given the following goal:

$$(Temperature > 30) \rightarrow (DYTune_K(N37) > 4)$$

N37 is a volume tuner and has an initial state 2. Variable "Temperature" records the temperature of the house, which can be easily embedded into the CSP planner. The states that if the temperature rises above 30, N37 needs to be tuned to more than 4.

Given the current temperature to be 32. The following shows the plan:

Case 4

Approach N37⇒ Tune up N37 by 1⇒ Tune up N37 by 1⇒ Tune up N37 by 1

Planning time = 0.25 s

Since the temperature is more than 20, N37 needs to be more than 4. With an initial value of 2, the robot needs to find ways to realize this with the activities it has. The robot will first approach N37, and then uses 1-increment tuner activity three times to get N37 to 5.

Currently, from literature review, there is no approach that can accommodate planning and number manipulation and reasoning under the same declarative language, except introducing an extension to it. To be able to describe them in one declarative way has two advantages, which are (1) Work on generalization of robot planning can be done (2) Optimization method can be easily studied to improve planning.

Case 5: Reusability of activities

As stated in the introduction, robot is useful in the evolution of the smart home, where it can dispense services that are yet to be supported by available smart devices. Since the number of smart devices will continue to grow, the robot is required to co-operate with these devices.

The services of these smart devices can be easily constructed with the precondition/effect definition [15], which can be used by the CSP planner. In this case, we consider the door to bedroom 2 (N15) to be installed a motor, which will open/close the door automatically. The activity definition is very simple, where it doesn't have a precondition, and the effect is just $DYStateNO_t(N15) = Open$ or $DYStateNO_t(N15) = Close$.

Lets consider the goal where the robot needs to move M3 (which is in bedroom 2) to the living room table (N34) as follows:

$$DYAtMO_K(M3) = N34$$

The generated plan is as follows:

Case 5

N15 opens (automatic)⇒ Pass door N15 to bedroom 2⇒ Approach N12⇒ Open N12⇒ Pick up M3 from N12⇒ Pass door N15 to the living room⇒ Approach N34⇒ Place M3 at N34

Planning time = 0.73 s

As shown in the plan, the activity for the automatic door can simply be executed to aid the robot. This can be performed without having to redefine sub-goals or ontologies, or making additional rules regarding additional alternatives. CSP planner can execute them where it sees fit, which shows the benefit of reusability.

Case 6: Complex goals

Case 6 showcase the ability to handle complex goals. In Case 1, goals are considered direct commands from the user. Yet in a lot of cases, goals are imposed by environmental situations and implicit rules, such as the opening/closing of the appropriate doors to cut off rooms to prevent smoke from spreading, turning on the right amount of light when the surveillance camera is activated, and so on.

For this case, assume the user wants to transfer a cold drink out of the fridge (N22) while he/she is attending something else in the wet kitchen, and that there is an inherent rule where the fridge must always be stocked with canned drinks. Besides, there is a little smoke in the

wet kitchen, which triggers a goal to request for window in the wet kitchen to be opened. The goals, both from human commands and trigger are listed below:

Goal 1:

$$\exists x, y((FMOType(x) = CanDrink) \wedge (DYAtMO_0(x) = N22) \wedge (DYAtMO_K(x) = y) \wedge (FNOType(y) = Cabinet) \wedge (DYStateNO_K(N22) = Close) \wedge (DYStateNO_K(y) = Close))$$

Goal 2:

$$\exists z((FMOType(z) = CanDrink) \wedge (DYAtMO_K(z) = N22))$$

Goal 3:

$$\exists x((FNOType(x) = Window) \wedge (DYStateNO_K(x) = Open) \wedge NOLocation(x, WetKitchen))$$

The first goal states that there is a canned drink,x ,that should be initially in the fridge and at the final stage, should be in an object, y, that is of type 'Cabinet'. This y should be closed at the final stage, and, like wise for the fridge. The second goal states that there is a canned drink, z, which should be in the fridge at final stage. The third goal states that an object x which is of type 'Window' and located in the Wet Kitchen should be opened. This goal is the smoke triggered goal.

The goals require the planner to make choices on the objects it need to fetch as well as the windows it needs to open, which demonstrates implicit goals.

The following shows the plan:

Case 6

Approach N19⇒ Open N19⇒ Pass door N19 to the kitchen⇒ Approach N22⇒ Open N22⇒ Pick up M1 from N22⇒ Approach N20⇒ Place M1 at N20⇒ Close N20⇒ Pick up M2 from N20⇒ Close N20⇒ Approach N22⇒ Place M2 at N22⇒ Close N22⇒ Approach N23⇒ Open N23⇒ Pass door N23 to the wet kitchen⇒ Approach N29⇒ Open N29

Planning time = 82.6 s

Robot execution is shown in Fig. 6. The robot will enter the kitchen and approach the fridge(N22) to obtain the cold drink (M1). It then chooses to place it in the kitchen cabinet(N20) due to the fact that all movable objects should be placed at a non-movable object like the cabinet. Objects will not be placed at non movable objects like doors as restrictions are made. Since N20 has another canned drink inside, the robot will transfer that to the fridge, after which all cabinet and fridge doors are closed. The robot then proceeds to the wet kitchen to open the window N29.

Although the goals are achieved, the time it took for planning is more than 1 min, which is considered too long, especially in times of emergency. Therefore, methods to distribute the goals into manageable sub-goals are required. But care should be taken as this may introduce Sussman Anomaly. For example, if Goal 1 and 2 are separated, the achievement of Goal 1 (bringing out M1 from the fridge) may be undone when trying to fulfill Goal 2 separately.

Besides, knowledge base can be further exploited by reducing as much uncertainty as possible. The use of existential quantifiers will increase search space, thus, slowing down the planning process. With more uncertainty as to which object to deal with, use of existential quantifiers can be reduced.

We would like to think of the intelligence of smart home to consist of the planner (as what this paper is about) and knowledge base (which consists of database and inference engine). As shown in this paper, the CSP planner is capable of performing inference to choose the appropriate object to attend to, yet, such inference can be made separately from the knowledge base. Therefore, there is a gray area of who should be dealing with that. In this paper, we are not ready to answer this question as it depends on the full design of the smart home, but the implication can be seen from how the goal is constructed as in Case 6. Given the three goals above (which are implicit), the planner needs to perform inference to choose the objects. But the three goals above can also be simplified to explicit direct goals, such as:

$(DYAtMO_K(M1) = N20)$, which means canned drink M1 should be in cabinet N20 as goal.

To have such simplified goal, knowledge base needs to come up with direct answers of what objects to attend to. This also means the making of choices is separated from the planner, and other complications might ensue. If the inference engine is run according to OWL description logic, there is yet another issue of it taking on the open world assumption, which requires further work to combine them.

Therefore, one can use solely the planner by providing it with well-constructed implicit goals (but may be complicated), and which, may take up more planning time, or, the knowledge base can come provide direct answers to the choices, but risk further complications due to different logic used.

Conclusions

Service composition provides robot with a higher level plan of execution. Service composition for a service robot is developed via representing and solving planner problem in terms of CSP. CSP provides a means to define problems declaratively, and is also able to support variables of wider domain. The few basic type of services and their corresponding terms are optimized for faster planning. Simulation shows that the system is able to perform tasks bounded by complex logical rules, yet no

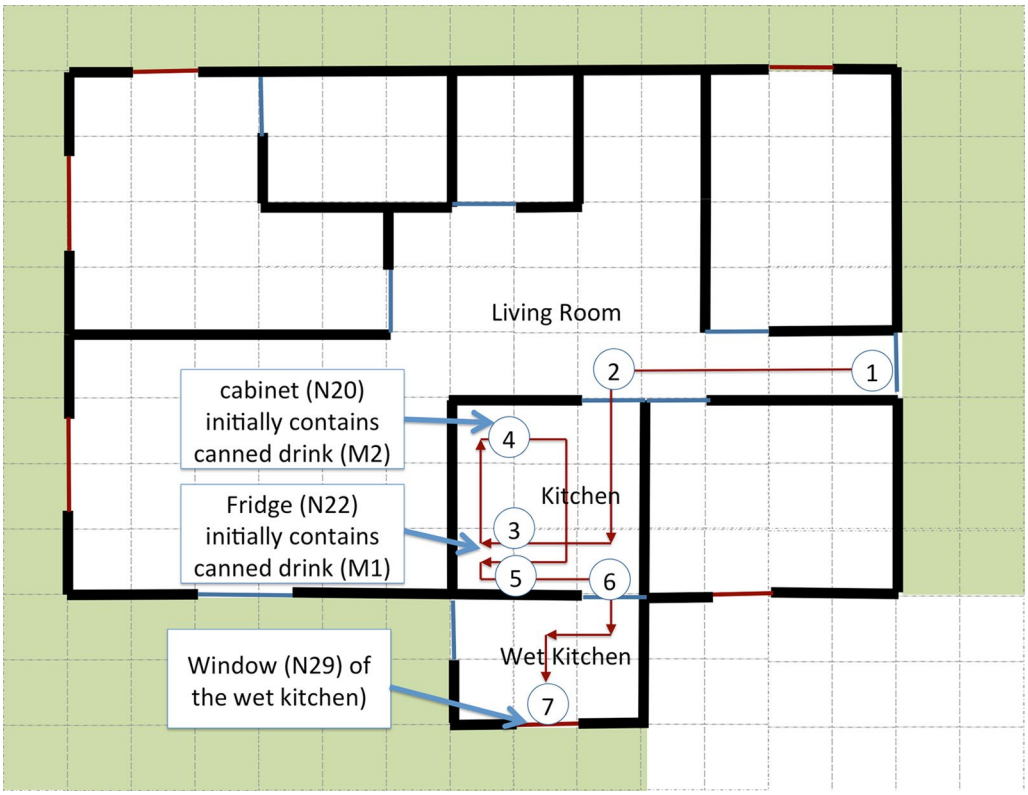

Fig. 6 Case 6: Complex goals The robot proceeds to the kitchen at Step 2 and goes toward the fridge(N22) to fetch a canned drink(M1) in Step 3. It subsequently goes to the cabinet(N20) to place M1. N20 contains another canned drink(M2). The robot will fetch M2 and close N20 at the end of Step 4. It proceeds to the fridge to place M2 and closed it in Step 5. After that, the robot approaches the wet kitchen in Step 6 and opens the window in Step 7

pre-programming is required to combine different services as they are loosely bounded. Intelligent decision making for object selection is embedded into the planning problem through the extended structure of the original CSP planner [15]. The method is also standardized to determine sub-locations for the sake of mobility, and also relationships between objects as well as their properties. This enables automatic generation of planning rules.

Currently, the system is far from perfect. Given the number of rules imposed without restriction, planning time will increase until it is not practical for emergency situation. Method to distribute goal into sub-goals is crucial in this respect. Besides, currently, all the constraints imposed are hard constraints. More research need to be done to determine the possible soft constraints that can be utilized for service robot.

In future work, knowledge base represented according to an appropriate building ontology should be exploited to provide much more intelligence towards determining the values of the terms. The addition of Description Logic to model home situation and their relationships can provide all necessary information to generate conditions of the services, while at the same time the robot will provide feedbacks back to the knowledge base.

Authors' contributions

NT develops the planner system and drafted the manuscript. AS implements the robot simulator and its connection with the planner. JB is involved in discussion of result and manuscript drafting. NK supervises the project. All members are involved in checking and approval of the paper. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Received: 9 January 2016 Accepted: 16 July 2016
Published online: 26 August 2016

References

1. Tang D, Yusuf B, Botzheim J, Kubota N, Chan CS (2015) A novel multimodal communication framework using robot partner for aging population. Expert Syst Appl 42(9):4540–4555
2. Chernbumroong S, Cang S, Atkins A, Yu H (2013) Elderly activities recognition and classification for applications in assisted living. Expert Syst Appl 40(5):1662–1674. doi:10.1016/j.eswa.2012.09.004

3. Broekens J, Heerink M, Rosendal H (2009) Assistive social robots in elderly care: a review. Gerontechnology 8(2):94–103
4. Yaguchi H, Sato K, Kojima M, Sogen K, Takaoka Y, Tsuchinaga M, Yamamoto T, Inaba M (2014) Development of 3d viewer based teleoperation interface for human support robot hsr. ROBOMECH J 1(1):1–12
5. Domingos P, Lowd D (2009) Markov logic: an interface layer for artificial intelligence. Syn Lect Artif Intell Machine Learn 3(1):1–155
6. Klusch M, Gerber A, Schmidt M (2005). Semantic web service composition planning with OWLS-XPLAN. In: Proc of the 2005 AAAI fall symposium on semantic web and agents
7. Hatzi O, Vrakas D, Bassiliades N, Anagnostopoulos D, Vlahavas I (2010) Semantic awareness in automated web service composition through planning. Lect Notes Comput Sci 6040:123–132
8. Hatzi O, Vrakas D, Nikolaidou M, Bassiliades N, Anagnostopoulos D, Vlahavas L (2012) An integrated approach to automated semantic web service composition through planning. IEEE Trans Serv Comput 5(3):319–332
9. Georgievski I, Aiello M (2014) An overview of hierarchical task network planning
10. Sirin E, Parsia B, Wu D, Hendler J, Nau D (2004) HTN planning for web service composition using SHOP2. Web Semantics Sci, Serv Agent WWW 1(4):377–396
11. Au TC, Kuter U, Nau D (2005) Web service composition with volatile information. In: Gil Y, Motta E, Benjamins VR (eds) The semantic web ISWC 2005. Lecture notes computer science. Springer-Verlag, Heidelberg, pp 52–66
12. Yang F, Khandelwal P, Leonetti M, Stone P (2014) Planning in answer set programming while learning action costs for mobile robots. In: AAAI Spring 2014 symposium on knowledge representation and reasoning in robotics (AAAI-SSS)
13. Zhang S, Sridharan M, Wyatt JL (2015) Mixed logical inference and probabilistic planning for robots in unreliable worlds. IEEE Trans Robot 31(3):699–713
14. Kaldeli E, Warriach EU, Bresser J, Lazovik A, Aiello M (2010) Interoperation, composition and simulation of services at home. In: Maglio P, Weske M, Yang J, Fantinato M (eds) Service-oriented computing. Lecture notes in computer science. Springer-Verlag, Heidelberg, pp 167–181
15. Kaldeli E, Warriach EU, Lazovik A, Aiello M (2013) Coordinating the web of services for a smart home. ACM Trans Web 7(2):10–11040
16. Kaldeli E, Lazovik A, Aiello M (2011) Continual planning with sensing for web service composition. In: Proc of 25th AAAI conference on artificial intelligence
17. Helmert M (2009) Concise finite-domain representations for PDDL planning tasks. Artif Intell 173(5):503–535
18. Richter S, Westphal M (2010) The LAMA planner: guiding cost-based anytime planning with landmarks. J Artif Intell Res 39(1):127–177
19. Hoffmann J, Weber I, Kraft F (2010) Sap speaks PDDL. In: 24th national conference of the American Association for artificial intelligence
20. Han SN, Lee GM, Crespi N (2014) Semantic context-aware service composition for building automation system. IEEE Trans Indus Inform 10(1):752–761
21. Allemang D, Hendler J (2011) Semantic web for the working ontologist: effective modeling in RDFS and OWL. Morgan Kaufmann Publishers, Burlington
22. Rao J, Küngas P, Matskin M (2006) Composition of semantic web services using linear logic theorem proving. Inform Syst 31(4):340–360
23. De Moura L, BjØrner N (2008) Z3: An efficient SMT solver. In: International conference on tools and algorithms for the construction and analysis of systems. pp. 337–340