An Ontology- and Constraint-based Approach for Dynamic Personalized Planning in Renal Disease Management

Normadiah Mahiddin[1], Yu-N Cheah[2], Fazilah Haron[3]
[1]Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600 Bangi, Malaysia
[2, 3]School of Computer Sciences, Universiti Sains Malaysia, 11800 USM Penang, Malaysia
[3]College of Computer Science and Engineering, Taibah University, P.O. Box 30002, Madinah, Saudi Arabia

***Abstract*—Healthcare service providers, including those involved in renal disease management, are concerned about the planning of their patients' treatments. With efforts to automate the planning process, shortcomings are apparent due to the following reasons: (1) current plan representations or ontologies are too fine grained, and (2) current planning systems are often static. To address these issues, we introduce a planning system called Dynamic Personalized Planner (DP Planner) which consists of: (1) a suitably light-weight and generic plan representation, and (2) a constraint-based dynamic planning engine. The plan representation is based on existing plan ontologies, and developed in XML. With the available plans, the planning engine focuses on personalizing pre-existing (or generic) plans that can be dynamically changed as the condition of the patient changes over time. To illustrate our dynamic personalized planning approach, we present an example in renal disease management. In a comparative study, we observed that the resulting DP Planner possesses features that rival that of other planning systems, in particular that of Asgaard and O-Plan.**

Keywords-patient care planning; treatment protocols; dynamic treatment planning; personal health services.

I. INTRODUCTION

Healthcare service providers are undoubtedly concerned about updating their patients' health records or profiles, and the planning of their patients' treatments to support the efficient and effective delivery of healthcare services. However, not all healthcare service providers are carrying out planning activities effectively, especially when it comes to automated or computer-based planning, due to shortcomings in current planning systems.

The first problem is that most of the current plan representations or ontologies are too fine grained (detailed). This means that the plan representations or ontologies are not suited for all situations and for all levels. We need to have a portable and intuitively easy representation that facilitates the storage and manipulation of generic plans. The second problem is that current planning systems are often static. This means planning is carried out once without taking into account changes that may take place as time goes on. These plans also do not consider past events. Dynamic planning is therefore required to allow plans to be updated as new situations arise.

To address the concerns above, we present a methodology for generic and dynamic healthcare planning, resulting in a system called the Dynamic Personalized Planner (DP Planner) [1]. For this purpose, we define (1) a suitably light-weight and generic plan representation based on existing plan ontologies, and (2) a constraint-based dynamic planning engine.

II. STATE OF THE ART

A popular approach for plan representation is via ontologies, i.e. plans are designed based on project specific ontologies and domain description languages [2]. Fig. 1 shows the structure of the Plan Ontology proposed by Tate [3].

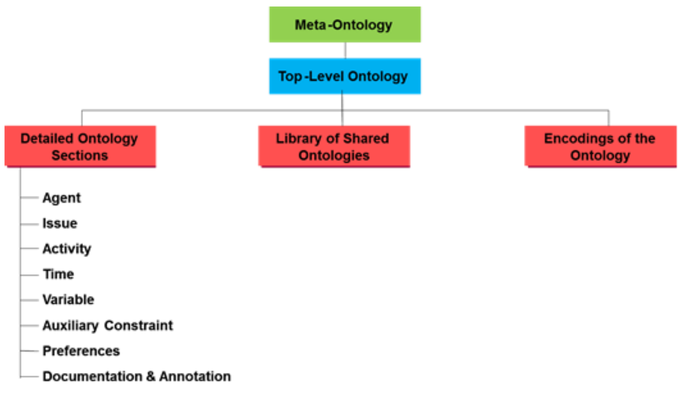

Meta-Ontology
Top-Level Ontology
Detailed Ontology Sections
Library of Shared Ontologies
Encodings of the Ontology
Agent
Issue
Activity
Time
Variable
Auxiliary Constraint
Preferences
Documentation & Annotation

Figure 1. Plan Ontology structure [3].

Another useful plan ontology for generic planning is the task-specific ontology and approach called Asbru which uses skeletal plans [4]. Asbru represents skeletal plans using a task-specific, intention-based, and time-oriented language. It can be used to design specific plans [5]. In Asbru, a plan contains a name and a set of arguments (time annotation and knowledge roles). All plans and actions have a temporal dimension and the plan's execution is controlled by a number of conditions such as filter, setup, suspend, reactivate, abort and complete [6]. Fig. 2 shows the Asbru plan ontology structure.

Besides plan ontologies (which contribute towards plan representation), there are also a number of plan generation initiatives. An example of such an initiative is the RAX Planner/Scheduler (RAX-PS) [7]. The RAX-PS generates plans that are temporally flexible, allowing the execution system to adjust to actual plan execution conditions without breaking the plan. The result is a system capable of building concurrent plans.

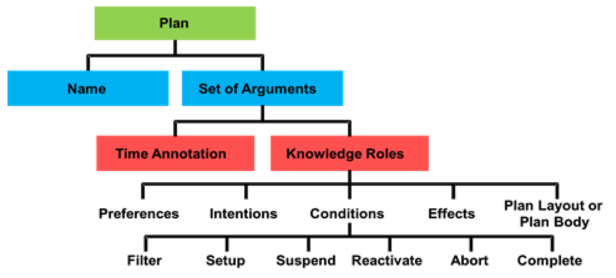


Figure 2. The Asbru plan ontology structure [4].

Fig. 3 shows the architecture of the planning system in RAX-PS. For the RAX-PS, the domain model describes the dynamics of the system [8] to which the planner is being applied, i.e. the Deep Space One Spacecraft. The plan database is initialized by a plan request which consists of an initial state and a set of goals. The initialized database is then modified by a search engine to generate a complete and valid plan. This complete plan is then put into operation by an execution agent. The heuristics and planning experts are also integral parts of the Deep Space One planning system. The heuristics guide the search engine while the planning expert interfaces with external systems which provide critical inputs such as altitude and speed.

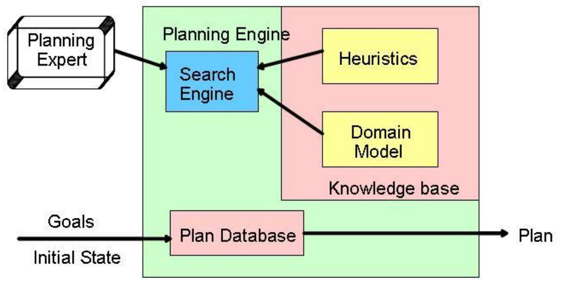


Figure 3. The RAX Planner/Scheduler (RAX-PS) [7].

The Capture the Flag (CTF) [9] game project uses the notion of critical points (time during the execution of an action or plan where a decision might be made) to define states in the continuous domain. These states are then used to efficiently evaluate plans. An action or a plan posts a goal, G. This invokes the CTF planning algorithm [10].

In the effort to generate outputs that are dynamically assembled from smaller fragments, the Personalized Healthcare Information (PHI) system [11] composes personalized documents that conform to an individual's health profile. The composition of PHI is carried out in a three-step procedure which are (1) selection of a set of Topic Specific Documents (TD), where each selected TD addresses some of the individual's healthcare concerns, (2) combination of the selected TDs to produce an aggregated PHI document, and (3) verification of the accuracy of the aggregated PHI document. Each individual illness/concern/issue noted in the health profile is addressed by at least a single TD. Constraint satisfaction techniques are used to ensure that the aggregated PHI document is medically consistent. Fig. 4 shows the processes for PHI composition.

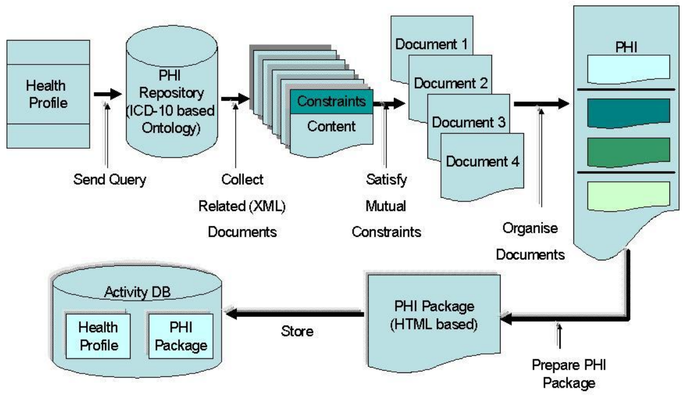


Figure 4. The process flow for PHI composition [11].

We have found the PHI system's approach in using constraints attractive as this approach could be adapted in our DP Planner to ensure the coherency of plan fragments that are assembled to form a complete plan.

III. The Dynamic Personalised Planning Methodology

The dynamic personalized planning approach consists of two phases:

1. Plan ontology definition and representation.
2. Planning algorithm definition.

Fig. 5 briefly shows the two phases together with the techniques and approaches used.

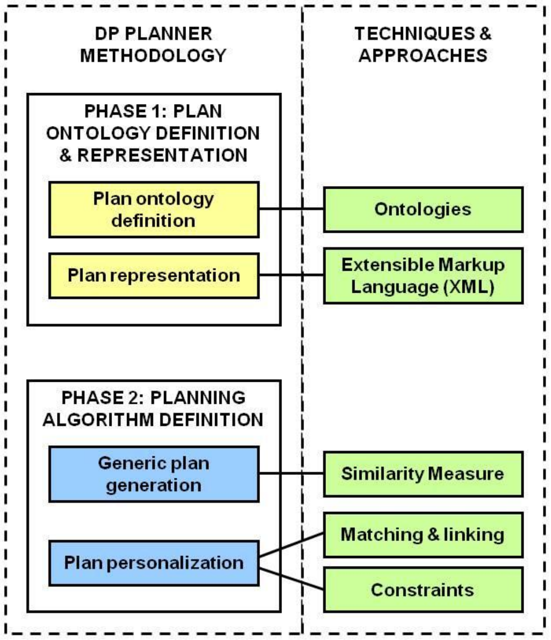


Figure 5. DP Planner methodology with related techniques and approaches [1].

A. *Phase 1: Plan Ontology Definition and Representation*

In defining the plan ontology, the Asbru plan ontology was adopted as the basis for the DP Planner's plan ontology in view that it is reasonably concise compared to other ontologies that were surveyed. Besides this, some elements of Goals, Operators, Methods and Selection (GOMS) analysis [12] were also incorporated. GOMS is a method for analyzing and modeling the knowledge and skills that a user must develop in order to perform tasks on a device or system.

1) *Plan ontology definition*

Fig. 6 shows the DP Planner's plan ontology [13].

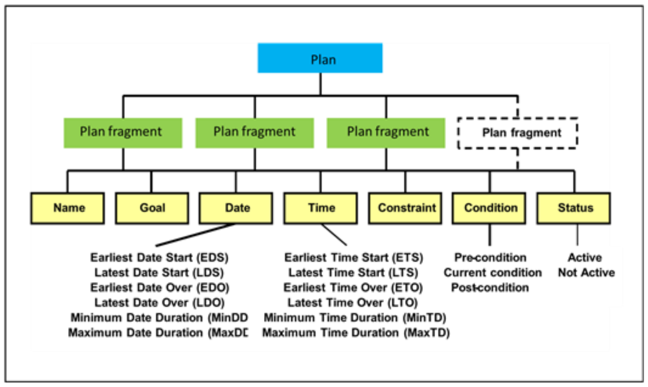


Figure 6. DP Planner's plan ontology structure [13].

Referring to Fig. 6, the plan is positioned at the highest position in the plan hierarchy, and is basically the task that needs to be performed. Examples of plans are those for kidney patient treatment monitoring, gestational diabetes mellitus monitoring, student performance monitoring, etc. The plan consists of a sequence of plan fragments. These plan fragments are the necessary steps to achieve the task and can be viewed as the most crucial component of the plan ontology.

Each plan fragment consists of seven attributes:

- Name: Identifies a plan.
- Goal: States the target to be achieved.
- Date: States the date of plan execution (if required).
- Time: States the duration of plan execution.
- Constraints: Information of the plan execution limit.
- Condition: Situations in which the task takes place.
- Status: Keeps track of the situation of plan execution.

Some of the plan component's attributes have detailed sub-attributes. Examples are as follows:

- Date has six sub-attributes: Earliest Date Start (EDS), Latest Date Start (LDS), Earliest Date Over (EDO), Latest Date Over (LDO), Minimum Date Duration (MinDD) and Maximum Date Duration (MaxDD),
- Time also has six sub-attributes: Earliest Time Start (ETS), Latest Time Start (LTS), Earliest Time Over (ETO), Latest Time Over (LTO), Minimum Time Duration (MinTD) and Maximum Time Duration (MaxTD).
- Condition consists of three sub-attributes:
    - Pre-condition: This is to ensure that certain conditions are met before the execution of a particular plan fragment.
    - Current condition: This ensures that certain conditions are currently met in a particular plan fragment.
    - Post-condition: This is to ensure that certain conditions are met after the execution of a particular plan fragment and before the next plan fragment.

2) *Plan representation*

Fig. 7 illustrates an example of a plan represented in XML for the treatment of a patient with renal disease. Note that the plan ontology can be naturally implemented in XML as the hierarchical nature of ontologies maps very well into the nested nature of XML. With the defined DP Planner plan ontology and representation, the DP Planner planning engine implementation is discussed next.

```
<?xml version="1.0" encoding="UTF-8"?>
  <PlanRepository>
    <Patient IC="781002046086">Ani</Patient>
    <Plan ID="1">
      <PlanFragment ID="1.1" Name="Glomerular Filtration (GFR) Test">
        <Goal>Detect the stage of Kidney Disease</Goal>
        <Date>
          <EarliestDateStart>20/4/2006</EarliestDateStart>
          <LatestDateStart>21/4/2006</LatestDateStart>
          <EarliestDateOver>00/00/0000</EarliestDateOver>
          <LatestDateOver>00/00/0000</LatestDateOver>
          <MinimumDateDuration>0 years 0 months 1 days</MinimumDateDuration>
          <MaximumDateDuration>0 years 0 months 1 days</MaximumDateDuration>
        </Date>
        <Time>
          <EarliestTimeStart>08:00 AM</EarliestTimeStart>
          <LatestTime_Start>04:00 PM</LatestTimeStart>
          <EarliestTimeOver>09:00 AM</EarliestTimeOver>
          <LatestTimeOver>05:00 PM</LatestTimeOver>
          <MinimumTimeDuration>0 hours 30 minutes</MinimumTimeDuration>
          <MaximumTimeDuration>9 hours 0 minutes</MaximumTimeDuration>
        </Time>
        <Condition>
          <PreCondition>Diabetes</PreCondition>
          <PreCondition>High Blood Pressure</PreCondition>
          <CurrentCondition>Tiredness</CurrentCondition>
          <CurrentCondition>Nausea</CurrentCondition>
          <PostCondition>Percentage of Renal Damage = 60%</PostCondition>
        </Condition>
        <Constraints>Not taken any medications</Constraints>
        <Constraints>Not pregnant</Constraints>
        <Status>Completed</Status>
      </PlanFragment>
    </Plan>
  </PlanRepository>
```

Figure 7. Plan representation in XML format.

B. *Phase 2: Planning Algorithm Definition*

The DP Planner's planning strategy involves reusing plans that are stored in the plan repository and subsequently personalizing these plans according to the constraints defined by the user.

The planning algorithm is divided into two parts (see Fig. 8):

1. Generic plan generation.
2. Plan personalization.

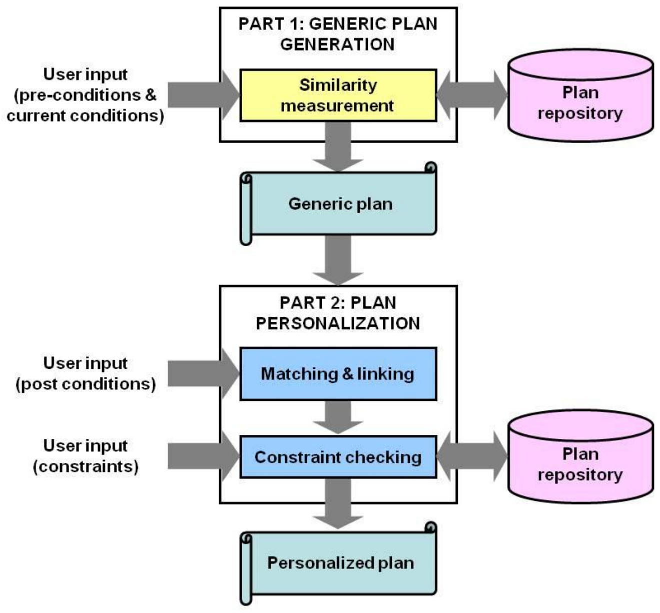


Figure 8. The planning algorithm.

1) Generic plan generation

Firstly, user inputs are matched against each of the existing plans in the plan repository. A concentration unit is calculated during the matching process to compare the closeness of the match. This unit is based on the number of matches between the user's inputs (pre-conditions and current conditions) with those of the plan fragments which constitute each plan. The similarity between the user's pre-condition and current condition inputs, and those of a particular plan, *P*, is expressed as C_P, (see Equation 1).

$$CP = \frac{n}{T_f} \quad (1)$$

where n = number of matches of pre-conditions + matches of current conditions, and T_f = total number of plan fragments in a plan. The values of C for each plan will be compared. The plan with the highest value will be chosen as the generic plan. After a generic plan has been identified, the process of plan personalization will follow

2) Plan personalization

The personalization method employs a combination of (1) a simple matching and linking technique, and (2) a constraint-based approach to form certain restrictions [9] so that only plan fragments that meet the predetermined criteria and user's inputs can form the finalized and personalized plan for a particular user or situation. Personalization is only carried out when the user's status is active, i.e. the plan is still relevant to the user's condition. For a start, the user will be asked for the outcomes of following the activities defined in a particular plan fragment. These outcomes are called post-conditions. Here, a simple matching technique will be applied to match the post-conditions of a particular (current) plan fragment with the pre-conditions in the next plan fragment. If these match each other, that next plan fragment in the generic plan will be recommended as the subsequent plan for the user. However, if they do not match, the plan (or sequence of plan fragments) will terminate at that current plan fragment.

The matching technique is applied to ensure that each plan fragment in the finalized plan links to each other (see Fig. 9). This is to ensure that the final plan generated by the system corresponds to the needs of the user. After ensuring that the plan fragments in the generic plan fulfils the initial matching of post-conditions with the pre-conditions, the actual personalization can then take place using constraints.

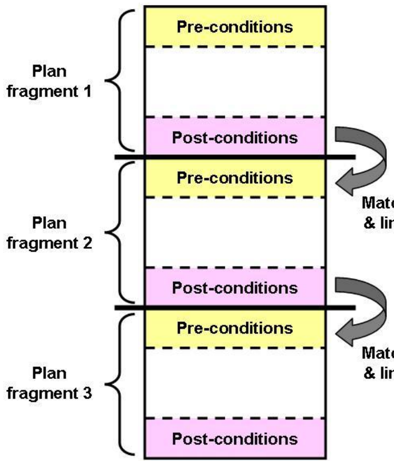


Figure 9. The linking between plan fragments in a plan.

Constraints are utilized to ensure the consistency of the multiple fragments of a plan in order to form a complete plan. The individual plan fragments must not contradict each other or lead to improper recommendations [11]. Therefore, for the purpose of the DP Planner, a constraint is simply a variable which restricts the execution of a particular plan fragment. It also describes the relationship between plan fragments in a plan. Ultimately, the constraints are used to find suitable replacements for plan fragments which are not suited for the plan fragments preceding or following it. Fig. 10 illustrates how constraints are used to personalize a generic plan.

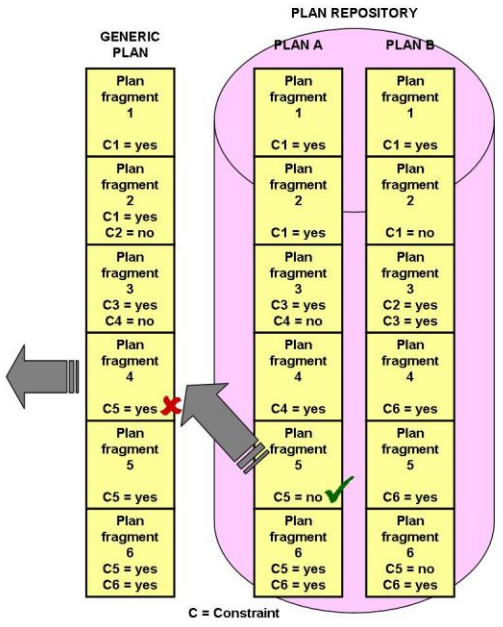
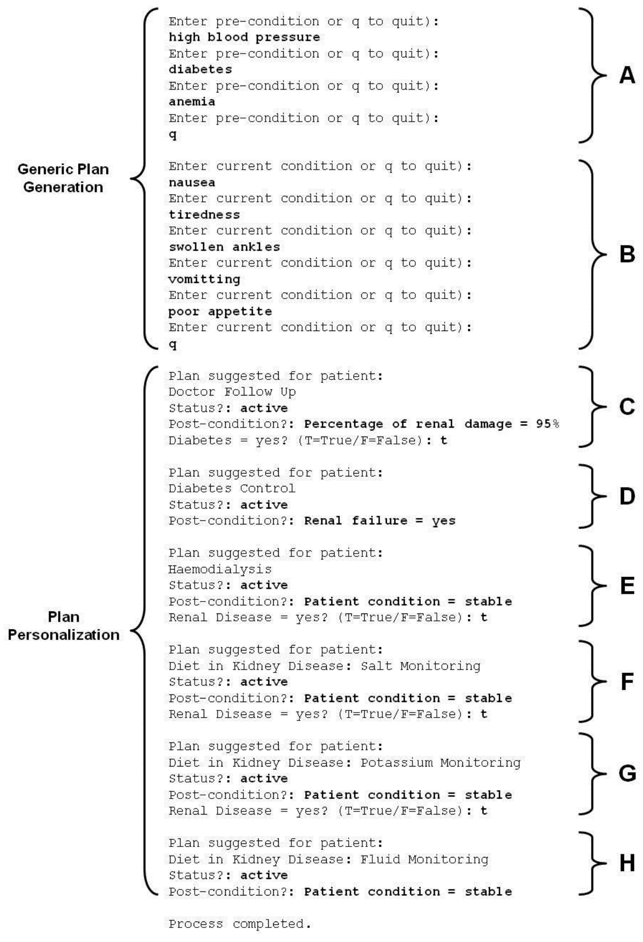


Figure 10. The use of constraints in plan personalization.

In Fig. 10, the generic plan's Plan Fragment 4 shows constraint 5 (C5) = yes. However, let us assume this does not fulfill a constraint specified by the user, i.e. the user has C5 = no. Therefore, plan personalization is carried out to find a plan fragment in the plan repository which fulfils the user's constraint. The example in Fig. 10 shows Plan Fragment 5 in Plan A has C5 = no. Therefore, our system will use this plan fragment to replace Plan Fragment 4 in the generic plan provided other details are also met (e.g. pre-conditions, post-conditions, etc.).

IV. Results: Example Cases

Fig. 11 shows the command-line system interface which obtains inputs from the user to generate a generic and personalized plan in the domain of renal disease.



```
Enter pre-condition or q to quit):
high blood pressure
Enter pre-condition or q to quit):
diabetes
Enter pre-condition or q to quit):
anemia
Enter pre-condition or q to quit):
q

Enter current condition or q to quit):
nausea
Enter current condition or q to quit):
tiredness
Enter current condition or q to quit):
swollen ankles
Enter current condition or q to quit):
vomitting
Enter current condition or q to quit):
poor appetite
Enter current condition or q to quit):
q

Plan suggested for patient:
Doctor Follow Up
Status?: active
Post-condition?: Percentage of renal damage = 95%
Diabetes = yes? (T=True/F=False): t

Plan suggested for patient:
Diabetes Control
Status?: active
Post-condition?: Renal failure = yes

Plan suggested for patient:
Haemodialysis
Status?: active
Post-condition?: Patient condition = stable
Renal Disease = yes? (T=True/F=False): t

Plan suggested for patient:
Diet in Kidney Disease: Salt Monitoring
Status?: active
Post-condition?: Patient condition = stable
Renal Disease = yes? (T=True/F=False): t

Plan suggested for patient:
Diet in Kidney Disease: Potassium Monitoring
Status?: active
Post-condition?: Patient condition = stable
Renal Disease = yes? (T=True/F=False): t

Plan suggested for patient:
Diet in Kidney Disease: Fluid Monitoring
Status?: active
Post-condition?: Patient condition = stable

Process completed.
```

Figure 11. The process of finding the best-matched plan in the plan repository.

A. Generic Plan Generation

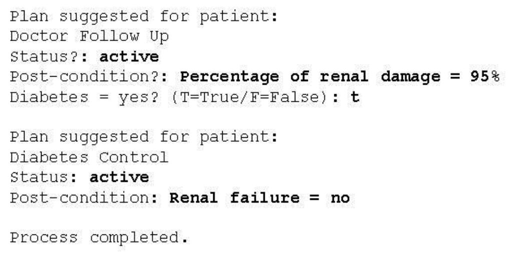
From Fig. 11, the interaction labeled A gets the inputs of pre-conditions while the interaction labeled B gets the inputs of current conditions from the user (bold text indicates text entered by the user). Both of these conditions will be used to match the pre-conditions and current conditions in the plans found in the plan repository. As described in Section III.B.1, this matching is carried out to find the plan that best matches the user's pre-condition and current condition inputs, i.e. the plan with the highest concentration of matches value (see Equation 1) is identified. Then, a copy of that plan is created as an XML file and assigned with a new ID. The values for the pre-conditions and current conditions are assigned with those inputted by the user while dates, times, and post-conditions are left empty.

B. Plan Personalization

After the generic plan has been generated, personalization then takes place. Firstly, the user will be presented with the first plan fragment in the plan as shown in Fig. 11 (labeled C). Then, the planning system will prompt the user about their status, i.e. whether active (user is implementing the plan) or not active (not implementing the plan). Either response will result in a different output for the user. Hence, we show the results of different cases in the following sub-sections. This process is repeated with other plan fragments (labeled D to H in Fig. 11).

1) Case 1: Status is active

This case is for situations when the user is implementing a particular plan fragment. When this happens, the system will prompt for details about the user's post-condition as shown in Fig. 12. Assuming that the user's input for post-condition was *Percentage of renal damage = 95%* therefore, this input will be matched with the next plan fragment's pre-condition. If they match, the constraints of the next plan fragment will be highlighted to the user: *Diabetes = yes?: (T=True/F=False)* as shown in Fig. 12. If the user fulfils the constraint, the next plan fragment will be presented. The output for this user will be generated as shown in Fig. 13.

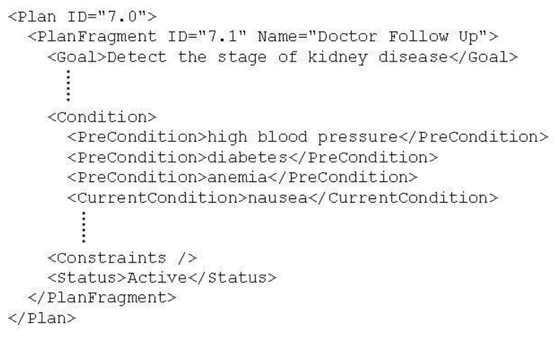


```
Plan suggested for patient:
Doctor Follow Up
Status?: active
Post-condition?: Percentage of renal damage = 95%
Diabetes = yes? (T=True/F=False): t

Plan suggested for patient:
Diabetes Control
Status: active
Post-condition: Renal failure = no

Process completed.
```

Figure 12. Part of the system interface when status of the user is "active".



```
<Plan ID="7.0">
  <PlanFragment ID="7.1" Name="Doctor Follow Up">
    <Goal>Detect the stage of kidney disease</Goal>
      ⋮
    <Condition>
      <PreCondition>high blood pressure</PreCondition>
      <PreCondition>diabetes</PreCondition>
      <PreCondition>anemia</PreCondition>
      <CurrentCondition>nausea</CurrentCondition>
        ⋮
    <Constraints />
    <Status>Active</Status>
  </PlanFragment>
</Plan>
```

Figure 13. Part of the system output when status of the user is "active".

2) Case 2: Status is not active

This case shows that the user is not implementing a particular plan fragment. When this is indicated by the user, the system will not perform any personalization in the subsequent plan fragments. Therefore, the planning process is deemed to be complete. The output for the user will be generated as shown in Fig. 14. Here, the result is personalized by removing the subsequent plan fragments that are not necessary.

```
<Plan ID="8.0">
  <PlanFragment ID="8.1" Name="Doctor Follow Up">
    <Goal>Detect the stage of didney disease</Goal>
    <Date>
      ⋮
    </Date>
    <Time>
      ⋮
    </Time>
    <Constraints />
    <Condition>
      <PreCondition>diabetes</PreCondition>
      <CurrentCondition>nausea</CurrentCondition>
      <PostCondition />
    </Condition>
    <Constraints />
    <Status>Not active</Status>
  </PlanFragment>
</Plan>
```

Figure 14. Part of the system output when status of the user is “not active”.

3) Case 3: Inputs do not fulfill conditions or constraints in the generic plan

This situation can be further divided into three sub-cases as follows:

1. Inputs which do not fulfill conditions in the generic plan.
2. Inputs which do not fulfill some conditions but fulfill constraints in the generic plan.
3. Inputs which do not fulfill constraints in the generic plan.

In the first case, consider the example system interface in Fig. 15. Assume that the post-condition in the generic plan is *Percentage of renal damage = 95%*, and this does not match with the user’s input which is *Percentage of renal damage = 60%*. Therefore, the personalization process is terminated and the entire planning process is deemed to have completed. As a result, the planning system generates the output as shown in Fig. 16 as the personalized plan for this case.

```
Enter pre-condition or q to quit):
high blood pressure
Enter pre-condition or q to quit):
diabetes
Enter pre-condition or q to quit):
q

Enter current condition or q to quit):
nausea
Enter current condition or q to quit):
vomitting
Enter current condition or q to quit):
swollen ankles
Enter current condition or q to quit):
tiredness
Enter current condition or q to quit):
q

Plan suggested for patient:
Doctor Follow Up
Status?: active
Post-condition?: Percentage of renal damage = 60%

Process completed.
```

Figure 15. System interface for inputs which do not fulfill conditions in the generic plan.

```
<Plan ID="9.0">
  <PlanFragment ID="9.1" Name="Doctor Follow Up">
    <Goal>Detect the stage of kidney disease</Goal>
    <Date>
      ⋮
    </Date>
    <Time>
      ⋮
    </Time>
    <Constraints />
    <Condition>
      <PreCondition>high blood pressure</PreCondition>
      <PreCondition>diabetes</PreCondition>
      <CurrentCondition>nausea</CurrentCondition>
      <CurrentCondition>vomitting</CurrentCondition>
      <CurrentCondition>swollen ankles</CurrentCondition>
      <Current_Condition>tiredness</Current_Condition>
      <PostCondition>Percentage of renal damage = 60%</PostCondition>
    </Condition>
    <Constraints />
    <Status>Not active</Status>
  </PlanFragment>
</Plan>
```

Figure 16. Personalized plan for inputs which do not fulfill conditions in the generic plan.

In the second case, consider the example system interface in Fig. 17. Let us assume that the post-condition in the generic plan, i.e. *Patient condition = stable* (see Fig. 18), does not match the user’s input which is *Patient condition = not stable* (Fig. 17). Therefore, the process terminates. As a result, the planning system generates the personalized plan as showed in Fig. 19.

```
Enter pre-condition or q to quit):
high blood pressure
Enter pre-condition or q to quit):
diabetes
Enter pre-condition or q to quit):
q

Enter current condition or q to quit):
nausea
Enter current condition or q to quit):
vomitting
Enter current condition or q to quit):
swollen ankles
Enter current condition or q to quit):
tiredness
Enter current condition or q to quit):
q

Plan suggested for patient:
Doctor Follow Up
Status?: active
Post-condition?: Percentage of renal damage = 95%
High blood pressure = yes? (T=True/F=False): t

Plan suggested for patient:
High Blood Pressure Control
Status?: active
Post-condition?: Renal failure = yes
Fistula operation = yes? (T=True/F=False): t

Plan suggested for patient:
Haemodialysis
Status?: active
Post-condition?: Patient condition = not stable

Process completed.
```

Figure 17. System interface for inputs which do not fulfill some conditions but fulfill constraints in the generic plan.

In the third case, consider the example system interface in Fig. 20. Let us assume that the user does not fulfill the constraint of *High blood pressure = yes*, the planning system would search the plan repository for a plan fragment which fulfils the user’s input. This plan fragment is then retrieved and used to replace the one in the generic plan which did not fulfill the user's input. Fig. 21 shows the personalized plan for this case.

```
<Plan ID="10.0">
  <PlanFragment ID="10.1" Name="Doctor Follow Up">
      ⋮
  <PlanFragment ID="10.2" Name="High Blood Pressure Control">
      ⋮
  <PlanFragment ID="10.3" Name="Haemodialysis">
      ⋮
    <Condition>
      <PreCondition>Renal failure = yes</PreCondition>
      <CurrentCondition>Patient condition = not stable</CurrentCondition>
      <PostCondition>Patient condition = stable</PostCondition>
          ⋮
  <PlanFragment ID="10.4" Name="Diet in Kidney Disease: Salt Monitoring">
      ⋮
</Plan>
```

Figure 18. Generic plan for inputs which do not fulfill some conditions but fulfill constraints in the generic plan.

```
<Plan ID="10.0">
  <PlanFragment ID="10.1" Name="Doctor Follow Up">
    <Goal>Detect the stage of kidney disease</Goal>
    <Date>
      ⋮
    <Time>
      ⋮
    <Condition>
      <PreCondition>high blood pressure</PreCondition>
      <PreCondition>diabetes</PreCondition>
      <CurrentCondition>nausea</CurrentCondition>
      <CurrentCondition>swollen ankles</CurrentCondition>
      <CurrentCondition>tiredness</CurrentCondition>
      <CurrentCondition>vomitting</CurrentCondition>
      <PostCondition>Percentage of renal damage = 95%</PostCondition>
    </Condition>
    <Constraints />
    <Status>Active</Status>
  </PlanFragment>
  <PlanFragment ID="10.2" Name="High Blood Pressure Control">
    <Goal>To reduce and delay the kidney damage</Goal>
    <Date>
      ⋮
    <Time>
      ⋮
    <Condition>
      <PreCondition>Percentage of renal damage = 95%</PreCondition>
      <CurrentCondition>tiredness</CurrentCondition>
      <CurrentCondition>nausea</CurrentCondition>
      <CurrentCondition>vomitting</CurrentCondition>
      <CurrentCondition>swollen Ankles</CurrentCondition>
      <PostCondition>Renal failure = yes</PostCondition>
    </Condition>
    <Constraints>High blood pressure = yes</Constraints>
    <Status>Active</Status>
  </PlanFragment>
</Plan>
```

Figure 19. Personalized plan for inputs which do not fufill some conditions but fulfill constraints in the generic plan.

```
Enter pre_condition or q to quit):
diabetes
Enter pre_condition or q to quit):
q

Enter current condition or q to quit):
nausea
Enter current condition or q to quit):
vomitting
Enter current condition or q to quit):
swollen ankles
Enter current condition or q to quit):
q

Plan suggested for patient:
Doctor Follow Up
Status?: active
Post_condition?: Percentage of renal damage = 95%
High blood pressure = yes? (T=True/F=False): f

Suggested treatment plan:
Glomerular Filtration (GFR) Test

Process completed.
```

Figure 20. System interface for inputs which do not fulfill constraints in the generic plan.

```
<Plan ID="11.0">
  <PlanFragment ID="11.1" Name="Doctor Follow Up">
    <Goal>Detect the stage of kidney disease</Goal>
    <Date>
      ⋮
    <Time>
      ⋮
    <Condition>
      <PreCondition>diabetes</PreCondition>
      <CurrentCondition>nausea</CurrentCondition>
      <CurrentCondition>vomitting</CurrentCondition>
      <CurrentCondition>swollen ankles</CurrentCondition>
      <PostCondition>Percentage of renal damage = 95%</PostCondition>
    </Condition>
    <Constraints />
    <Status>Active</Status>
  </Plan_Fragment>
  <PlanFragment ID="11.2" Name="Glomerular Filtration Test">
    <Goal>To reduce and delay the kidney damage</Goal>
    <Date>
      ⋮
    <Time>
      ⋮
    <Condition>
      <PreCondition>Percentage of Renal Damage = 95%</PreCondition>
      <CurrentCondition>nausea</CurrentCondition>
      <CurrentCondition>vomitting</CurrentCondition>
      <CurrentCondition>swollen ankles</CurrentCondition>
      <PostCondition>Renal failure = yes</PostCondition>
    </Condition>
    <Constraints />
    <Status>Not active</Status>
  </PlanFragment>
</Plan>
```

Figure 21. Personalized plan for inputs which do not fulfill constraints in the generic plan.

V. Discussion and Comparison

In general, the generic and personalized plan generation processes performs up to expectation. However, as a limitation of the DP Planner system, these processes would not function fully when a replacement cannot be found in the plan repository. When the system encounters this situation, it will advise the user to refer the case to a medical practitioner.

TABLE I. Comparison of DP Planner with Other Planning Systems.

	Features/Planning system	DP Planner	Asbru/ Asgaard	O-Plan	I-X	Prodigy	STRIPS	PLANET	RAX-PS
Plan ontology & representation	Adoption of a conceptual model for planning	Yes	Yes	No	No	No	No	No	No
	Simplicity of the plan ontology	Yes	Yes	Yes	Yes	No	No	No	No
	Simplicity of the plan representation	Yes	Yes	No	Yes	Yes	Yes	No	No
Planning method	Planning approach	Non-linear	Non-linear	Non-linear	Non-linear	Non-linear	Linear	-	Non-linear
	Genericity	Domain-Configurable	Domain-Configurable	Domain-Configurable	Domain-Configurable	Domain-Configurable	Domain-Independent	-	Domain-Specific
	Application of case-based plan adaptation techniques	Yes	Yes	No	No	No	No	-	No
	Tolerance towards planning system execution failures	Yes	Yes	Yes	Yes	Yes	No	-	No

Table 1 shows the comparison between the DP Planner with other planning systems, i.e. Asgaard, O-Plan, Prodigy, STRIPS, PLANET and RAX-PS. From our observation, Asgaard and O-Plan are established planning systems that the DP Planner can be compared to in view that they have the relevant plan ontology, plan representation, as well as the planning engine for their planning system. Further comparisons with Asgaard and O-Plan are discussed in the following sub-sections.

A. *Plan Ontology Definition and Representation*

Asgaard was inspired by Belief-Desire-Intention (BDI) model [14] while DP Planner was developed based on Goals-Operators-Method-Selection (GOMS) model. Using the BDI framework, Asgaard has been used to build large-scale, highly capable agent system [15]. Therefore, Asgaard is more suited for domains with large and complex but partly vague and incomplete knowledge. In contrast, DP Planner is based on the GOMS framework that has not been used to develop large-scale systems. GOMS has been mainly used to represent human knowledge necessary for performing certain tasks and complex human activities. As a result, DP Planner which is based on GOMS is more suited in domains with obvious knowledge, i.e. knowledge that is confirmed and complete, for performing certain tasks and knowledge.

Due to its simplicity, the DP Planner plan ontology was developed without the need for any ontology tool such as Protégé. The DP Planner plan representation in XML is also intuitively easy to understand. Asbru (which is Asgaard's plan representation language) uses a machine-readable language (Backus-Naur Form or BNF syntax) to annotate guidelines based on the task-specific ontology.

Asbru also requires the use of an ontology editor such as Protégé for the acquisition of clinical guidelines based on the same ontology and GMT (Guideline Markup Tool) to translate the guideline into a formal representation written in XML [16].

DP Planner's ontology is also easier and simpler compared to O-Plan which has its own detailed ontology structure. The O-Plan plan representation is in Task Formalism form and will change in different O-Plan agents in which it is situated. This is quite complex compared to DP Planner in which the plan representation remains in XML form in any situation.

B. *Planning Algorithm Definition*

DP Planner generates a generic plan by retrieving an existing plan with similar characteristics to the current planning requirements, and adapting the generic plan by reusing existing plans to produce a personalized plan. This is akin to a case-based approach with adaptation. Asgaard employs a similar approach to DP Planner whereby it also applies plan adaptation in its planning process.

However, the difference is that Asgaard adopts the transformational type of adaptation whereas DP Planner adopts a derivational analogy type followed by the transformational type. Derivational analogy potentially reduces the search space by ignoring the unnecessary choices. This is achieved using the DP Planner's similarity measurement technique. This is only suitable in situations when most of the previous plans require extensive adaptation and when the cost of saving rationale is low [15]. The cost of saving rationale here means the ability to fulfill the requirements of a particular plan fragment that was defined as a constraint. However, it presupposes that the derivational traces exist. In contrast, when this is not possible, transformational analogy is the better choice because the plans themselves can be used for adaptation. Thus, in DP Planner, derivational analogy is applied in generic plan generation while transformational analogy is applied in the plan personalization in view that the cost of saving rationale is higher.

In general, Asgaard appears more robust in view that it is a fully deployed system, and that it has a monitoring component which monitors changes to the user's situation, while DP Planner is not a deployed system and therefore relies on users to report any changes.

O-Plan on the other hand is based on software agents and provides a hierarchical planning architecture to support planning and control with temporal and resource constraint handling [17]. O-Plan is also designed as a fully deployed system. O-Plan's architecture shows that it has five major components which are Domain Information, Knowledge Sources, Support Tools, Plan State, and Controller. O-Plan also has the agent architecture since it has a Task Assignment, Planner and Execution agent.

In O-Plan, its plan repair algorithm involves two tables (see Fig. 22): TOME (Table of Multiple Effect), and GOST (Goal Structure Table) [18]. An execution failure occurs when one or more of the expected effects at a node-end fail to be asserted. Each effect is recorded in the TOME and when an action depends on an effect asserted earlier, it is recorded in the GOST (Step 1).

When an execution failure occurs, the TOME will be updated and its relation with GOST entries will be found. If it is related with any of the GOST entries, then the Knowledge Sources is used to fix the problem (Step 2). The Knowledge Sources are responsible for determining the consequences of unexpected events or of actions that do not execute as intended, for deciding what action to take when a problem is detected, and for making repairs to the effected plan (Step 3 and 4) [17].

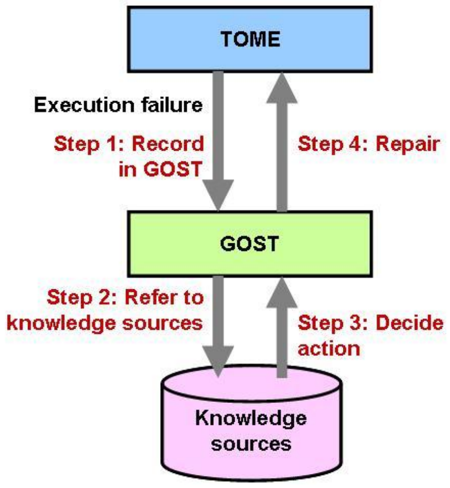


Figure 22. Solving execution failure in O-Plan [12].

When comparing the DP Planner's approach with that of O-Plan, it seems that the DP Planner approach is simpler as only two stages are needed to solve the failure whereas O-Plan requires four stages to solve the failure (see Fig. 23).

VI. Future Work

Presently, the DP Planner is implemented in a local environment. Its capabilities can be extended further by deploying it in a grid environment with distributed plan repositories and planning engines.

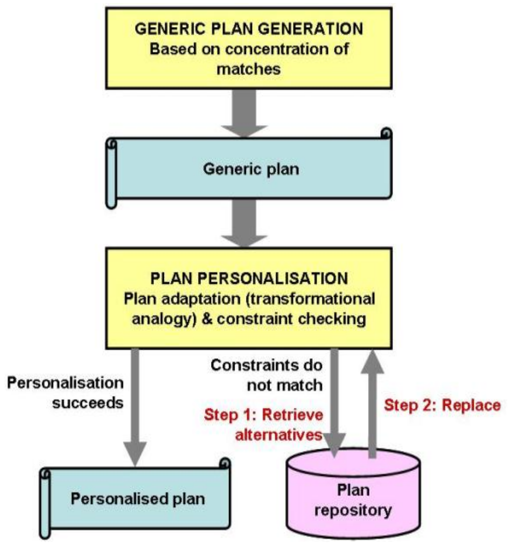


Figure 23. Solving execution failure in DP Planner.

For this purpose, the open source Globus Tool Kit can be utilized to allow sharing of computing power, databases and other tools online across corporate, institutional, and geographic boundaries autonomously and safely. The Globus Tool Kit is used in tandem with the Open Grid Services Architecture with Data Access and Integration (OGSADAI) which provides a service group registry that can be used to identify database services that offer specific data tables [19].

In order to communicate with OGSADAI, the DP Planner would potentially require middleware software to communicate with the DBMS which will store the planning data. Fig. 24 shows the OGSADAI with Globus in a possible DP Planner planning scenario. Here, the planning engines and plan repositories are distributed across different locations and each of these planning engines accesses the XML database containing the plan repository (or medical treatment plans) via Xindice (a suggested XML DBMS) while the planning results are delivered through the Client Tool Kit middleware whereby it provides the communication channel between the requesting node and the processing planning engine [20].

VII. Conclusion

There are many types of planning systems currently available in the literature though most are static in nature. In this paper, we presented (1) a plan ontology and representation, and (2) a dynamic planning engine which makes use of generic plans and plan fragments, based on our plan ontology, as a planning template. The processing of the planning engine is based on similarity measure, matching, linking, and constraint techniques. In the comparative study carried out, it was observed that the DP Planner possesses features that rival that of other planning systems, in particular that of Asgaard and O-Plan. It is hoped that the DP Planner will make planning initiative more efficient and effective in delivering applicable plans to users especially to healthcare providers and patients.

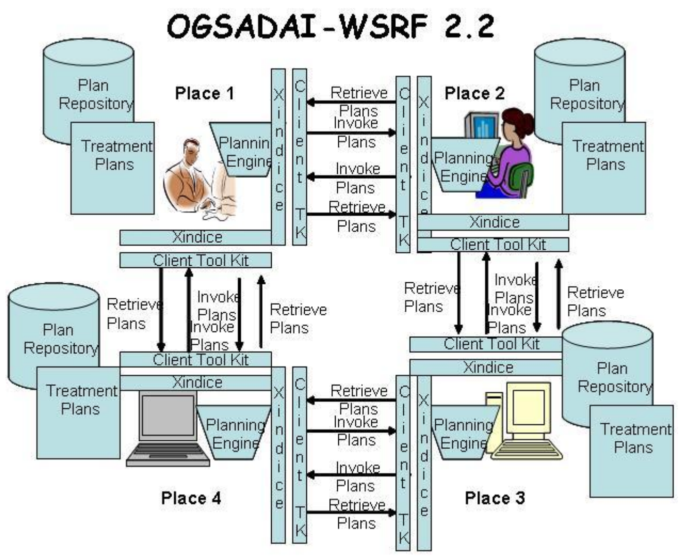


Figure 24. The DP Planner in a grid computing environment.

ACKNOWLEDGMENT

The authors wish to thank the Ministry of Higher Education, Malaysia and Universiti Sains Malaysia for funding this work through the Fundamental Research Grant Scheme (FRGS) under the project entitled "A Dynamic Planning Algorithm Using Ontologies and Constraints on the Grid".

REFERENCES

[1] N. Mahiddin, "An Ontology and Constraint-based Approach for Dynamic Personalised Planning," Master Thesis, Universiti Sains Malaysia, 2009.

[2] A. Tate, "Towards a plan ontology," Journal of the Italian Association for AI, AI*IA Notizie, Special Issues on Aspects of Planning Research, vol. 9, no. 1, pp. 19-26, March 1996.

[3] A. Tate, "A plan ontology - a working document," in Proceedings of the Workshop on Ontology Development and Use, La Jolle, CA, U.S.A., 1994.

[4] Y. Shahar, S. Miksch, and P. Johnson, "A task-specific ontology for design and execution of time-oriented skeletal plans," in Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1996.

[5] S. Miksch, Y. Shahar, and P. Johnson, "Asbru: a task specific, intention-based and time-oriented language for representing skeletal plans," in Proceedings of the Seventh Workshop on Knowledge Engineering Methods and Languages (KEML-97), Milton Keynes, U.K., 1997.

[6] S. Miksch, A. Seyfang, and R. Kosara, "Plan management: supporting all steps of protocol development and deployment," in Proceedings of the EUNITE-Workshop on Intelligent Systems in Patient Care, Vienna, Austsria, pp. 35-42, 2001.

[7] A. K. Jónsson, P. H. Morris, N. Muscettola, and K. Rajan, "Planning in interplanetary space: Theory and practice," in Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000), Breckenridge, CO, U.S.A., pp. 177-186, 2000.

[8] J. Frank, A. K. Jónsson, and P. H. Morris, "On reformulating planning as dynamic constraint satisfaction," in Proceedings of the 4th International Symposium on Abstraction, Reformulation and Approximation, London, U.K.: Springer-Verlag, 2000.

[9] M. S. Atkin, and P. R. Cohen, "Physical planning and dynamics," in Working Notes of the AAAI Fall Symposium on Distributed Continual Planning, Orlando, FL, U.S.A., pp. 4-9, 1998.

[10] M. S. Atkin, and P. R. Cohen, "Using simulation and critical points to defines states continous search spaces," in Proceedings of Simulation Conference, Orlando, FL, U.S.A., vol. 1, pp. 464-470, 2000.

[11] S. S. R. Abidi, Y. H. Chong, S. R. Abidi, "An intelligent info-structure for composing and pushing personalised healthcare information over the internet," in Proceedings of the 14th IEEE Symposium on Computer Based Medical Systems (CBMS 2001), Bethesda, MD, U.S.A., pp. 225-230, 2001.

[12] D. Jonassen, M. Tessmer, and W. Hannum, Task Analysis Methods for Instructional Design, Mahwah, NJ: Lawrence Erlbaum Associates, 1999.

[13] N. Mahiddin, Y.-N. Cheah, and F. Haron, "A generic plan ontology for dynamic health plans," in Proceedings of the International Conference of Knowledge Engineering 2005 (IKE '05), Las Vegas, NV, U.S.A., 2005.

[14] S. Miksch, and A. Seyfang, "Continual planning with time-oriented, skeletal plans," in Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000), Amsterdam, The Netherlands: IOS Press, pp. 512-515, 2000.

[15] H. M. Avila, and M. T. Cox, "Case-based plan adaptation: An analysis and review," IEEE Intelligent Systems, vol. 23, no. 4, pp.75-81, 2008.

[16] Y. Shahar, S. Miksch, and P. Johnson, "The Asgaard project: A task specific framework for the application and critiquing of time-oriented clinical guidelines," Artificial Intelligence in Medicine, vol. 14, pp. 29-51, 1998.

[17] O-Plan Team, O-plan: Architecture guide (version 2.3), 1995. URL: http://www.aiai.ed.ac.uk/oplan/documents/ANY/oplan-architecture-guide.pdf. Retrieved: 18 October 2011.

[18] B. Drabble, A. Tate, and J. Dalton, "Repairing plans on-the-fly," in Proceedings of the NASA Workshop on Planning and Scheduling for Space, 1997.

[19] Globus Project Team, Globus tool kit homepage, 2007. URL: http://www-unix.globus.org/toolkit/about.html. Retrieved: 18 October 2011.

[20] Globus Project Team, Software components for grid systems and applications, 2007. URL: http://www.globus.org/grid_software. Retrieved: 18 October 2011.

[21] Felix, A. A., & Taofiki, A. A. (2011). On Algebraic Spectrum of Ontology Evaluation. *IJACSA - International Journal of Advanced Computer Science and Applications*, *2*(7), 159-168.

[22] Vanitha, K., Yasudha, K., & Soujanya, K. N. (2011). The Development Process of the Semantic Web and Web Ontology. *IJACSA - International Journal of Advanced Computer Science and Applications*, *2*(7).

AUTHORS PROFILE

Normadiah Mahiddin received her B.Comp.Sc. (Hons) degree from Universiti Sains Malaysia in 2003, and M.Sc. (Computer Science) from the same university in 2009. She is currently a Ph.D. candidate at Universiti Kebangsaan Malaysia. Her research interests include knowledge management, intelligent systems, health informatics, and automated planning.

Yu-N Cheah received his B.Comp.Sc. (Hons) and Ph.D. degrees from Universiti Sains Malaysia in 1998 and 2002 respectively. He is currently lecturing at the School of Computer Sciences, Universiti Sains Malaysia. His research interests include knowledge management, intelligent systems, health informatics, and semantic technologies.

Fazilah Haron received her B.Sc. (in Computer Science) from the University of Wisconsin-Madison, U.S.A. and her Ph.D. from the University of Leeds, U.K. She is an Associate Professor at the School of Computer Sciences, Universiti Sains Malaysia and currently on secondment at Taibah University, Madinah, Saudi Arabia. Her research interests include modeling and simulation of crowd, parallel and distributed processing, and grid computing.