GoalMorph: Partial Goal Satisfaction for Flexible Service Composition

Maja Vukovic and Peter Robinson

Computer Laboratory
University of Cambridge
15 JJ Thomson Ave, Cambridge CB30FD, UK
{firstname.lastname}@cl.cam.ac.uk

Abstract: **AI Planning has proven to be a valuable tool for generating composite services in a range of application domains, such as travel planning and supply chain management. However, it can fail to satisfy a composite service request if one or more parts of the goal cannot be reached, due to the context changes or missing service descriptions. As goals are structured as conjunctions of goals states, each representing a partial solution, satisfying some goal states instead of all can be better than satisfying none of them. In this paper, we present GoalMorph, a framework for context aware goal transformation that (a) constructs context aware goals and (b) reformulates failed goals into problems that can be solved by the AI planner. The core of the approach is ContextMesh, a context ontology, which facilitates context layering – the process of expanding or reducing the number of context types and generating corresponding context goal transformations. Furthermore we employ a goal utility model that allows partial satisfaction and demonstrate how it can be exploited to generate a transformed goal. We discuss evaluation results, and demonstrate that our implementation provides a practical and scalable solution.**

***Keywords*:** Web service composition, AI planning, context-awareness, partial goal satisfaction**.**

I. INTRODUCTION

Web service composition, the process of assembling composite Web services from collections of individual, interoperating Web services, has recently gained much attention to support business-to-business or enterprise application integration as well as the development of context aware applications. It has been used to provide software solutions in many application domains ranging from lifecycle management[1], travel planning [2], making reservations for dining and movies [3, 4], content and news conversion services [5], etc. Available methods for composing Web services include rule-based systems [4], planning [6], view integration [7], scripting and coordination languages [8], to name a few.

We have previously presented a framework for context aware Web service composition [9], which uses the TLPlan [10] planner for the selection of the required services and sequencing of their execution. Different service compositions result from different contexts such as: resources available, time constraints, user profile and location.
Contextual changes may trigger further re-planning during the execution of the services, causing the application to evolve dynamically. This work is motivated by the requirements for fault-tolerant, context-aware service composition. Context aware service composition may fail if there are missing service descriptions, context changes, conflicts between goals or insufficient time/resources to solve the goal. The classical planning model requires the satisfaction of all goal states (conjunctions). In case one of the goals cannot be satisfied, the entire request fails, and that results in a lack of service provision to the user. Our system is based on the concept of partial goal satisfaction; in many cases it is better for the user to get a composite service that partially satisfies her goals than no composite service at all.

In this paper[1] we present a comprehensive context aware goal transformation framework GoalMorph, which transforms failed goals into the ones that can be solved. GoalMorph separates goal conditions depending whether they are arising from a user's request into – *core* goals or are side-effects of the current user's context into – *context* goals. The core component of our framework is a ContextMesh, a context ontology consisting of a number of hierarchies along which the context is organized – it specifies (application-specific) concepts of context types and their relationships. It plays the central role in transformation of goal by facilitating context layering – the process of context *folding* and *unfolding*, to expand or reduce the amount of contextual data and the corresponding goals that need to be satisfied.

GoalMorph makes the following main contributions:

1. A model for representation of context aware goals.
2. Analysis and taxonomy of core and context goal types and corresponding transformations.
3. A model of partial satisfaction to reason about partial satisfaction of core and context goals.
4. A utility model to reason about goal transformations and corresponding partial success in achieving one goal against partial success in achieving another goal.

This paper extends our previous work on context aware service composition to allow context aware (1) goal construction and (2) goal transformation. Section 2 provides background on planning technology and its application to Web service composition problem. We identify and examine the challenges in context-aware, planning-based, service composition by means of an example scenario in Section 3. Section 4 analyzes the architecture of our framework and its prototype implementation. Section 5 presents and discusses our evaluation results. Section 6 positions our work in the research context of partial goal satisfaction and goal-oriented

[1] This is the extended version of paper presented at NWESP 2005 [11].

service composition. We outline and discuss future work in Section 7 and conclude in Section 8.

II. BACKGROUND

Planning is a problem solving technique, where knowledge about available actions and their consequences is used to identify a solution over an abstract set of possible plans [12]. There are three main inputs to the planner:

1. Initial state: A description of the starting world state.
2. Goal state: A description of the desired world state.
3. Operators: A set of descriptions of operators that transform the world states

The output of the planning process is a plan—a sequence of actions that can be executed in order to achieve the desired goal. By explicitly declaring Web services as processes in terms of their inputs, outputs, preconditions and effects, we can apply the goal-oriented inferencing from the planning technologies for the Web service composition.

```
(:goal
  (and
  (direction_found)
  (direction_speech_out)))
```

Figure 1 Sample TLPlan goal definition for service composition request

Planning goals serve two purposes. Firstly, they represent information about the planning problem or composition request, by providing criteria for delivering a successful plan – which goals states must be satisfied. For example, a user's request for driving directions may be described by the conjuction of goal states in TLPlan syntax[2] is shown in Figure 1. Secondly, goals limit inference in the planning process, by allowing a planner to backchain the goal propositions.

III. RUNNING SCENARIO AND TECHNICAL CHALLENGES

To illustrate how context aware applications can be built as collections of cooperating services designed to interact with one another, we synthesize a suitable procedure for context aware restaurant lookup dynamically based on user location, activity, and computing device; ensuring that the resulting service looks and feels the same irrespective of the device they use.

Table 1 shows timelines of three different cases of this scenario. In the first case, Josh is using his laptop in the Lab in Cambridge, with subscription to the entertainment portal that provides restaurant recommendation service to make lunch plans with his College friends. The portal in turn uses UK-based restaurant and driving direction services to help him locate a Spanish restaurant that makes his favorite dish.
Later in the day, Josh is in Zurich and wishes to go to the closest Lebanese restaurant. The local restaurant guide service, however, provides the restaurant information only in German or French. Furthermore, it is formatted for presentation on a mobile phone. As Josh is driving, he would prefer the restaurant directions to be routed to his in-vehicle information system and delivered in speech synthesized form. A special new service for Josh may be assembled from component services: RestaurantFinder, DirectionsFinder, English translation and reformatting for in-vehicle information system appliances (i.e. text to speech translation).

A. *Explicit Representation of Context Aware Goals*

In all cases presented in Table 1, Josh's high level goal is retrieval of restaurant directions. However, this high level request maps to a different system goal in a different context. The planning goals are explicit descriptions of the goal state(s) to be reached. For instance, when Josh is driving, directions should be in the speech-synthesized form, which results in addition of a further goal condition (directions speech out), shown in Figure 1, as opposed to textual display when he is walking on the street. Conventional planning systems are designed to deal with explicitly defined goals. A common approach in service composition is the use of goal repositories, which store predefined, formalized goal descriptions. This method is rather impractical, embedding the dependencies between all possible combinations of context values and the corresponding goal conditions would make it difficult to extend later to take into account new context values and types. In any case it may be impossible to foresee all contexts in which the user may submit a request.

B. *Partial Goal Satisfaction*

Consider Case 3 of the scenario, where the planning process may fail for a number of reasons – the planner having wrong goal descriptions, the planner having incomplete knowledge of the domain, or unavailability of required operators (e.g. if there is no text-to-speech capability in this in-vehicle system).

This should not result in the unsuccessful termination of the composition process, as the approximation of the original goal may be satisfied, and a partial and viable solution still presented to Josh. For example, the system can detect the extended context information. By knowing that Josh is only driving at 20km/h, and is about to reach a red traffic light, it could be reasonably safe to deliver the directions to his SmartPhone – and therefore removing text2speech requirements from the original goal – still satisfying it partially. Furthermore, the system could also infer wider context in which Josh submitted his request for restaurant directions.
By observing his social setting and detecting he is with Anne, the system could forward turn-by-turn directions to her mobile phone, and she could guide him through the maze of central Zurich.

[2] Each goal is a defined either as the predicate or function symbol of the domain by the first-order formulas. The goal predicate definition consists of the name — name of the defined symbol (i.e. function, predicate or generator) and arity – that specifies the number of parameters accepted by the defined symbol. The corresponding goal instance then is described by the symbol name and arguments. Both goal literals in Figure 1 have 0 arity.

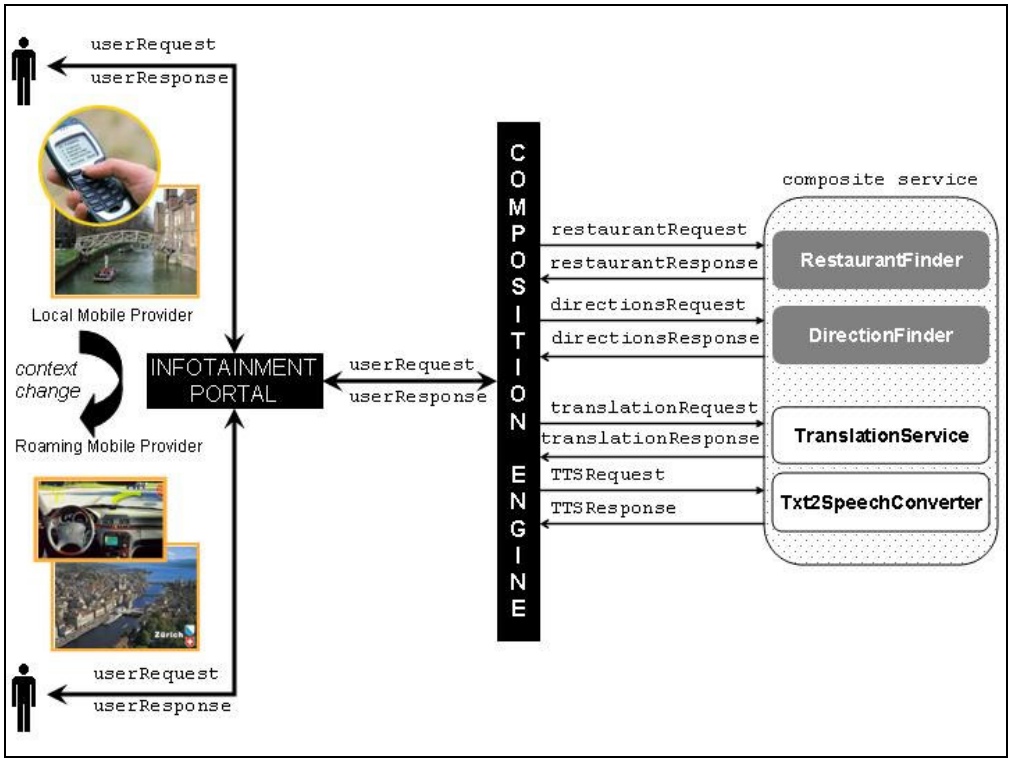


Figure 2 Context Aware Service Composition: Restaurant Finder Scenario

	Input: Context data				Output: Expected behavior	
Case	**Activity**	**Time**	**Device used**	**Location**	**Text to speech**	**Translation**
1	Sitting	10am	Laptop	Office, Computer Lab, Cambridge	N/A	N/A
2	Walking	12.30pm	SmartPhone	Home, Newnham, Cambridge	Read out directions	N/A
3	Driving	7pm	Embedded	In-vehicle, Limmatstrasse, Zurich	Read out directions	Translate to English

Table 1 Sample context and expected application behavior in context aware restaurant finder.

I. GoalMorph: Framework for Context Aware Goal Transformation

GoalMorph is a comprehensive context aware goal transformation framework or transforming goals into problems that can be solved by the planner. `GoalMorph` classifies goal conditions into core goals, which are arising from a user's request, and context goals, which are side-effects of the current user's context. `ContextMesh` is a context ontology consisting of a number of hierarchies along which the context is organized – it specifies (application-specific) concepts of context types and their relationships. It facilitates context layering – context folding and unfolding, the process of expanding or reducing of number of context types and the corresponding goals that need to be solved.

In this section we introduce our goal taxonomy, describe the main components of the context aware goal transformation framework, and the operations they provide.

A. Goal Taxonomy

A user's request for a context aware composite service consists of description of the user's computational task intention and the current context, such as user location, activity, and computing device as described in Section 3. Therefore goal literals, explicitly describing aspects of the service request, are either implied by user's task intention, such `(direction found)`, or by the context itself, such as `(directions speech out)` that is a result of the user's driving activity. Correspondingly we separate goals based on their origin, into intention driven—core goals and context driven ones —context goals. Our taxonomy of goal conditions consists of (1) core goals, (2) base core goals, (3) dependent context goals and (4) independent context goals, which together form the basis for our work on goal transformation to enable partial goal satisfaction.

Core Goal Any goal condition that purely describes the user's task intention is a core goal. In our scenario from Section 3 core goals would be `(restaurant_found)` and `(direction_found)` goal conditions, whereas `(restaurant_found)` is also a base core goal.

Base Core Goal The absolute minimal core goal condition that needs to be satisfied to achieve a viable solution is a base core goal. It may therefore not be removed from the goal definition. For example, in our scenario where Josh requests directions to the restaurant, the base goal is to find a restaurant. To fulfill the service request and supply the user with a viable solution this base goal or its respective transformation must be reached.

Dependent Context Goal A Context goal condition, which can be seen as the attribute of the goal condition (or directly related to it) is a dependent context goal. For example `(directions speech out)` the goal literal relies on the presence of (directions found) the goal literal. If the core goal is removed from the goal set, the corresponding dependent context goals are also removed. (For example, removal of `(direction_found)` implies removal of `(directions_speech_out)`.

Independent Context Goal A context goal condition that does not (necessarily) directly affect the user's request is considered to be an independent context goal. For example, in our scenario we may want to add the goal condition of lowering the volume level of an in-car audio system while presenting the driving directions using the text to speech interface.

B. GoalMorph: Framework Overview

The entry point in the `GoalMorph` system is a request for a composite service, as shown in Figure 4.The user selects the base of the composition request from the `GoalRepository` (*getRestaurantDirections* — Step 1 in Figure 4), which holds the available core goal templates, explicitly defined by domain engineers using the planning language (Step 3a). `ContextService` is a general middleware infrastructure for context collection from a large number of disparate sources and the dissemination of that information (Step 2) to interested clients. Our context service component uses the solution as suggested by Lei et al. [13] to retrieve context such as: user's location, device in use and user's activity. `ContextProxy` generates context goal conditions that constrain this composition request given the context information, provided by the `ContextService`.

The final composition request is finally assembled from the core goal conditions from `GoalRepository` (Step 3a) and context goal conditions (Step 3b) from `ContextProxy`. The planner, the core of the composition engine, takes the problem definition (Step 4a) and domain definition (Step 4b) and uses a knowledge about available actions and their consequences to identify a solution (i.e. plan—a sequence of actions that can be executed in order to achieve the desired goal). This attempt may fail when the precise goal can not be achieved or the domain knowledge is incomplete.

When planner failure occurs (Step 5b in Figure 4) control is passed to the Goal Transformation Engine that transforms the goal into a problem that can be solved by planner. Firstly, it attempts to transform core goal(s), to see if some form of the original goal can be satisfied, by interacting with `GoalRepository` (Step 6a).

```
(define
(problem demo_problem)
(:domain demo_restaurant)
(:objects
  r_type
  r_city
  c_address
  r_address
  r_name)
(:init
(and
  (restaurant_type r_type)
  (current_address c_address)
  (restaurant_name r_name)
  (restaurant_city r_city)
  (restaurant_address r_address)
  (restaurant_country_de)))
(:goal
  (and
  (direction_found))))
```

(a) Planning goal for Case 1, shown in Table 1

```
(define
(problem demo_problem)
(:domain demo_restaurant)
(:objects
  r_type
  r_city
  c_address
  r_addres
  r_name)
(:init
  (and
  (restaurant_type r_type)
  (current_address c_address)
  (restaurant_name r_name)
  (restaurant_city r_city)
  (restaurant_address r_address)
  (restaurant_country_ch)
  (activity_driving)))
(:goal
  (and
  (direction_found)
  (direction_speech_out)
  (direction_language_en)))
```

(b) Planning goal for Case 3, shown in Table 1

Figure 3 Explicit goal representation for cases 1 and 3 of the motivating scenario

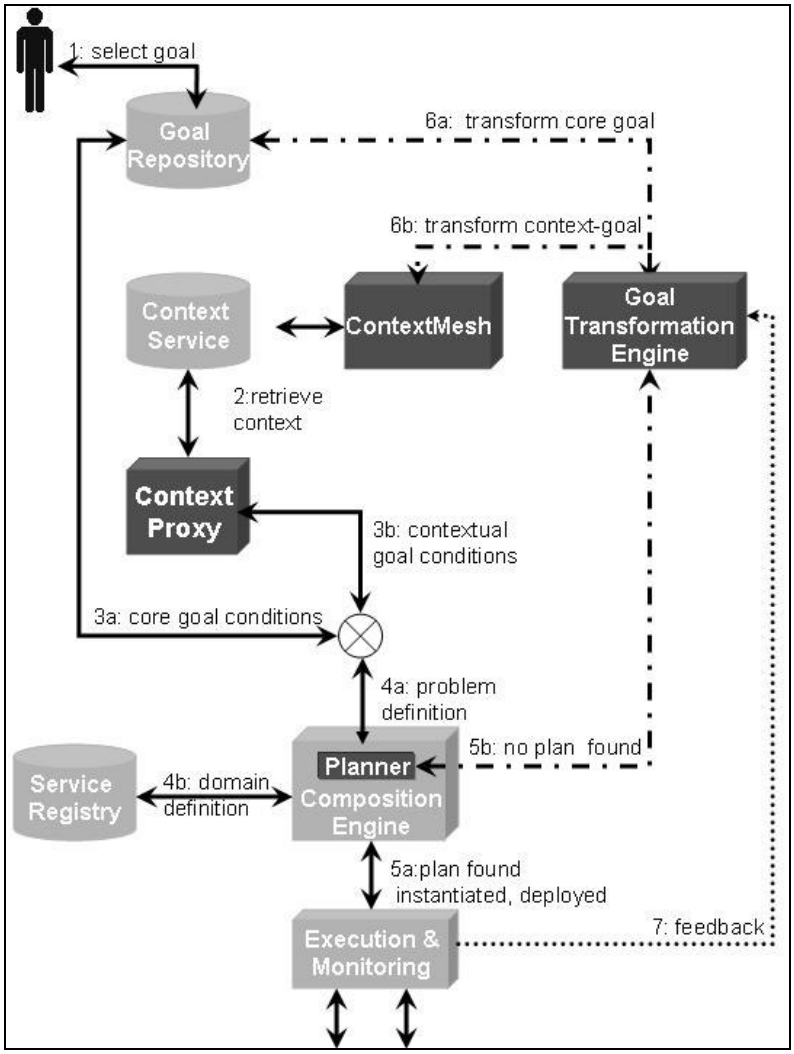


Figure 4 Framework for context aware goal transformation

```
;;goal predicate ;;sample utility
-------------------------------------------------------------------------------
(catering_facility_found cuisine_argument)                                  1
(catering_facility_take_away_found_found cuisine_argument)                  2
(catering_facility_home_delivery_found cuisine_argument)                    3
(catering_facility_take_away_found cuisine_argument)                        3
(catering_facility_eat_in_found cuisine_argument)                           4
(restaurant_found cuisine_argument)                                        10
(specialized_restaurant_found cuisine_argument)                            15
(catering_facility_bistro_found cuisine_argument)                           5
(catering_facility_cafeteria_found cuisine_argument)                        3
(catering_facility_cafe_found cuisine_argument)                             3
```

Figure 5 Sample Goal Type Hierarchy for Predicate (restaurant_found)

The transformed goal is then passed to the `ContextMesh` (Step 6b), which performs context layering: refining the context goal conditions by context unfolding or relaxing the context goal conditions by context folding. These operations are described in detail below. The context layering is guided by the context importance measures provided by the user through a GUI (Step 1). The transformed goal is then fed to the planner and the successful composition is evaluated by the user to refine the goal transformation utilities.

A. GoalRepository

The `GoalRepository` plays a central role in the creation of composite service requests. One obvious, but deficient way to solve service request generation is to have users manually select the individual goal conditions. Unfortunately, while the users may know what they want, they may not know how to realize it in a particular planning language. Requiring a user to understand the low-level details of an unfamiliar planning syntax, including the details of the domain knowledge is clearly unreasonable. Furthermore aside from the users (and their task intention), there are two

more key stakeholders in the service composition request: (1) service providers – service descriptions and their properties restrict the set of available goals and (2) context data.

Our goal taxonomy, presented in Section 4.1, separates goal conditions into core and context ones, based on their role in the composition request. Goal Repository is a software component that (1) stores the explicit core goal representations of supported users tasks and (2) contains an ontology of those explicit goals facilitating goal transformations.

Each goal is specified as a list of predicates and functions (goal conditions) that must be true in the goal world. For example, the high level task of Josh is "*Find the directions to the closest Spanish restaurant*". This task maps to the explicit goal request shown in Figure 3(a). The core parts of the goal description are the goal conditions (restaurant_found) and (direction_found). Furthermore, the (directions_speech_ out) goal literal is the context goal condition that arises due to the current user context (i.e. activity driving). The mapping from user's task description in natural, user-friendly, language "*Find the directions to the restaurant*" to goal declaration in the planning language is supplied by a system (domain) engineer.

In `GoalMorph`, goals are represented in Planning Domain Description Language (PDDL) [14]. PDDL is an action-centred language, inspired by the well-known STRIPS [15] formulations of planning problems. This allows for easy import of goals into the `GoalRepository`, and enables our framework to be used with any PDDL-based planner.

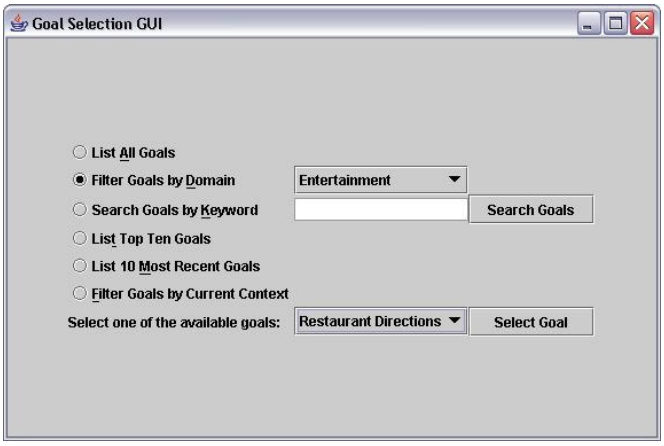


Figure 6 User Interface for Goal Selection

There are several ways for the user to select a goal, as shown on in Figure 6 representing the GUI for goal selection. Once the user selects the task, the corresponding goal is retrieved from the `GoalRepository`, and user is prompted to supply the necessary arguments to complete this composition request. `GoalRepository` keeps track of all the goal requests, their corresponding context, and number of their invocations, to enable automated goal selection (i.e. having the system actively making context-aware goal recommendations). The core goal from Goal Repository together with context goals obtained from `ContextProxy` form the composition request, which is passed to the planner. When the planner fails to devise a plan, a goal transformation is attempted to generate a goal that can be solved by the planner. For this purpose, the `GoalRepository` contains a goal ontology consisting of a number of hierarchies along which core goals are organized. Figure 5 shows ontology for the `(restaurant_found)` goal literal. With each goal condition we associate a utility value, which is used to select the goal conditions that contribute to the effectiveness of the overall service request when devising
a partial solution. For example, the goal literal with highest utility is (specialized restaurant found cuisine argument) 15. Goal utilities are application- and user- specific, and are synthetically assigned in the current implementation of the prototype. Issues about designing a system to create and maintain these utility values are also being considered (as a future work), however this paper concentrates on their use.
The `GoalRepository` provides a number of operations for navigation through goal hierarchies – along both type and argument dimension, which aid the substitution (or removal) of goal type or goal argument so that the new, transformed, goal may be solved.

Core goal transformations can take several forms:

Generalization: Obtain value of parent goal type / goal argument
Definition: Movement along goal type or argument hierarchy respectively towards the more generic value

Example 1: Type generalization
The (restaurant_found spanish) goal state may be substituted by the (catering_facility_found spanish) goal.

Example 2: Argument generalization
Similarly if the Spanish restaurant requirement from the previous example cannot be satisfied, trying to reach the goal of (restaurant_found mediteranean) or relaxing this argument (see below) may provide a partial solution.

Specialization: Obtain value of child goal type / goal argument
Definition: Movement along the goal type or argument hierarchy respectively towards a more specific value

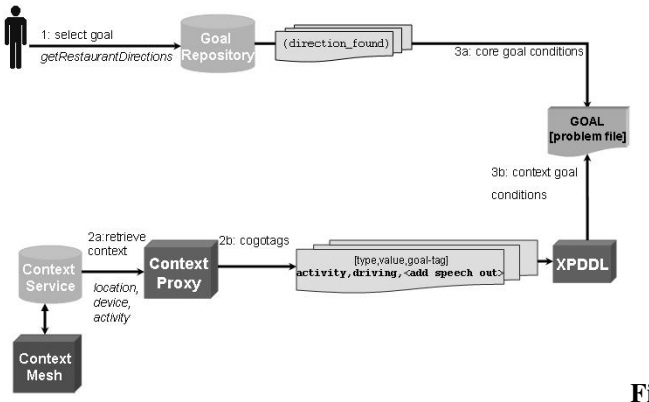
Example 1: Type specialization
The (restaurant_found spanish) goal state may be substituted by the (tapas_restaurant_found spanish) goal. Depending on the domain engineering, this categoriation of the restaurant can be also described by an additional argument such as (restaurant_found spanish tapas).

Example 2: Argument specialization
Once the base of the goal request is satisfied, we may want to impose further constraints on the goal. For instance, requesting that the Spanish restaurant found must have a parking facility.

Relaxation: Removal of goal type / goal argument
Definition: Removing the goal type or goal argument constraint

Example 1: Type removal
Removal of this goal condition from the set of goal states to be satisfied.

Example 2: Argument removal
Finding any restaurant, as opposed to the Spanish one.



Fi

Figure 7 ContextProxy, facilitating context dependent goals, using cogotags.

B. *ContextProxy: Generating Context Aware Goals*

To accommodate for the vast variety of – possibly even unanticipated – context types and their values that may be encountered and their impact on goal conditions, we introduce **cogotags** – **co**ntext **go**al **tags**. Each cogotag consists of the context type, value and the goal condition it introduces. Different context data may imply addition of goal conditions to the system or subtraction of goal conditions from the goal.

To facilitate publishing and making cogotags generally portable we present them in XML form, as shown in Figure 7.

Cogotags can either be provided by the user, context middleware, or inferred from the previous interaction of user with the system. XPPDL3[3] converts XML-based cogotags to a planner readable PDDL-based goal conditions.

```
<context-type> activity
</context-type>
<context-value> driving
</context-type>
<add-goal-condition> text2speech
</add-goal-condition>
```

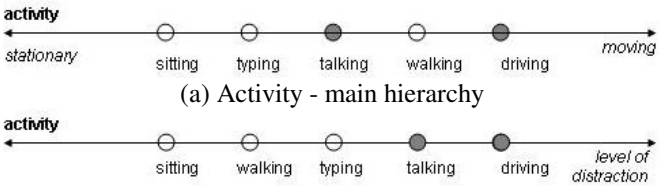
Figure 8 Sample cogotag

The `ContextProxy` is a component that interacts with context middleware and composition request manager. It assembles context goal conditions and passes them to the composition request manager, which together with core goal conditions form the final composition request. The context middleware an attach cogotags to the context data, otherwise the ContextProxy steps in and adds or removes goal conditions based on the user profile, or past interactions with the system.
When no context data is available (e.g. due to sensor failures) the `ContextProxy` fetches historical context from the `ContextService` using `ContextMesh`. Using cogotags removes the need for pre-built queries, and allows for flexible context goal generation, independent of context middleware used. We envisage users creating and carrying their own cogotags on the device or associated with personal profile. System can then learn from these and users' interactions with the system.

C. *ContextMesh*

The `ContextMesh` is a context ontology consisting of a number of hierarchies along which context provided by the Context Service is organized. It is a specification of the concepts of context types and their relationships existing in application domains. The `ContextMesh`, as an context ontology, can be equated with taxonomic hierarchies of context types, definitions, and the subsumption relation. `ContextMesh` organizes context along standard conceptual type-hierarchies within which instances are categorized.

For example, the activity of the user can be naturally organized by the fact whether user is stationary or moving, such as driving and typing can be specific instances of activity sitting (stationary) or walking and running can be instances of moving, shown in Figure 9(a).



(a) Activity - main hierarchy

(b) Activity - level of distraction

Figure 9 Two hierarchies for context activity

The ContextMesh allows for multiple (scenario-specific) hierarchies of each context type. Aside from its conventional and natural abstraction type-hierarchy, context types can be organized as an enumerated set, a numbered line, or a containment of components. The activity context type could be organized in the following manner: as a set of activities that can occur at a specific location; or along the numbered line according to the estimated duration of activity, and finally as a component partonomy – where activity can be organized in a graph, with the part-of relations.

For example, in our scenario the activity values may be further organized according to the level of distraction of the user. Figure 9(b) depicts how typing (i.e. working on the laptop) may be considered less distracting than driving.
Furthermore – context values are not mutually exclusive. For instance, Josh could very well be talking to the passenger (or on the cellphone) and driving the car at the same time, as shown by shaded values in Figure 9(b).

[3] http://www.cis.strath.ac.uk/ jg/XPDDL/

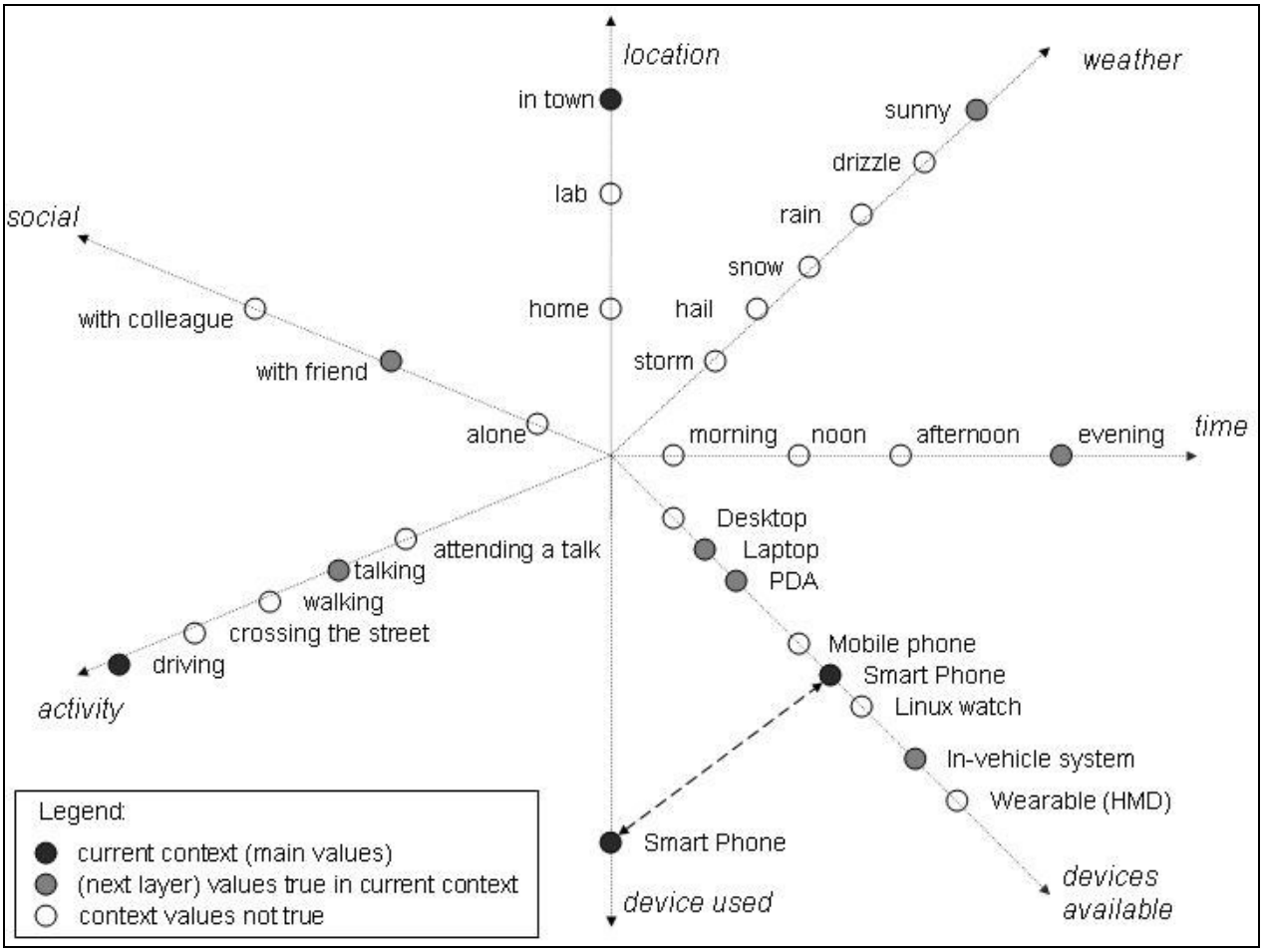


Figure 10 ContextMesh: Context cluster for restaurant finder.

```
<scenario restaurant>
<context-dimension activity>
<values ... >
<related-type location utility=1000 hierarchy=location_hierarchy1/>
<related-type time utility=500 hierarchy=time_hierarchy1 />
<hierarchy duration activity_hierarchy1 orderings ... />
<hierarchy distraction activity_hierarchy2 comparison_function.../>
</context-dimension>
...
</scenario>
```

Figure 11 Internal representation of context type and utilities

A number of interesting issues arise here. When selecting appropriate transformations, the first issue is which of these two activities are of higher relevance in the current scenario, which hierarchy should be used when determining goal transformations. One solution would be to have this importance measure specified at the domain engineering design stage. Secondly, how does the conjunction of more than one context value affect the anticipated behavior of the application.

For example, the conjunction of talking and driving is in itself more distracting than driving alone. In such a case, one would definitely want to avoid displaying or even reading out the driving directions.

Context is by nature highly interleaved. Activity, location, device used, physical environment, weather etc. are all inter-related. The relative importance of context types is overwhelmingly scenario specific. For example, consider the case where a user requests directions to a restaurant.
In this case location, activity and device used are of a higher importance than time requirements, weather conditions, lighting and noise. In contrast, primary context types for e-learning could be time requirements, user profile and device used.

One possible approach to managing all the context hierarchies and scenario-specific preferences is to introduce a utility function and associate it with each context type – thereby identifying that satisfying one context goal can be traded off against success in satisfying another context goal; and also that satisfying a context goal to a greater extent is preferable to satisfying it to a lesser extent. Again, we specify context utilities at the domain engineering stage and refine them through user feedback.

Correlation of different context types generates context clusters, which are scenario specific. They facilitate unfolding and folding of the context dimensions. In other words, context cluster controls the number of context types taken into account during the process of goal transformation.

Figure 10 shows the context cluster created for our scenario – a set of independent context axes. Initially only the device that the user is currently using is taken into account – a sold line. The current value of this context type, represented by fully filed circle, is SmartPhone. In the next iteration of context layering we incorporate information about the other available devices in the environment (initially an axis represented by dashed lines in Figure 10). Pervasive computing environments are characterized by richness of context, ContextMesh and the process of context layering therefore allow us to control the amount of context data passed to the system along with the service composition request.

1) ContextMesh Data Model and Management

The concepts and relationships in the ContextMesh are stored in XML format, as shown in Figure 11, overlaying the context data provided by `ContextService`. It therefore allows for any other context ontology to be imported and merged with it, provided that the necessary translation mechanism exists. In addition to providing the mechanism for merging other ontologies it is necessary that the context utilities and relations are defined to allow for goal transformation.

Every `ContextMesh` type dimension and corresponding hierarchies have an XML representation and they can be converted to ContextMesh object (dimension and/or hierarchy). Within our Java implementation ContextMesh hierarchies are represented by objects of the type ContextHierarchy. In particular ContextMesh pulls values of context type eligible for transformation by using getParent(), getSibling(), getChild() methods of a ContextHierarchy object, and receives values that are used to generate new context goal conditions.

Which ContextHierarchy should be used is scenario dependent, and is decided using the utilities. ContextHierarchy holds the comparison function according to which values are organized. ContextHierarchy implements a comparable interface, which is used to determine the order/hierarchy of the values stored in the ContextDimension, allowing the developer to define customized comparision of ContextHierarchy elements.

Along with each value a utility is stored for each hierarchy. This is used to access the context values stored in ContextDimension for this context type. Each context type (i.e. its ContextDimension) may be related to other context types. Corresponding ContextDimension objects store these relationships. For example activity may be associated with location and time of day. Each context relationship has a "utility" associated with it, which is used to determine the "strength" of relationships among different context type pairs. These relationships and their utilities are scenario-specific.

`ContextMesh` has methods for creating `ContextDimension` objects, which represent values of each context type, methods for adding and updating context hierarchies for a dimension, methods for importing new values of a context type, methods for managing relationships between context types, and methods for obtaining and creating a `ContextDimension` and `ContextHierarchy` objects from their XML representation.

2) Navigating through ContextMesh

ContextMesh provides the following operations for navigation through contextual data:

Context value specialization

Definition: Movement along the specified context type hierarchy towards a more general value (i.e. parent value)

Example: Goal to display information on LCD screen may be substituted by a goal to display on CRT.

Context value specialization

Definition: Movement along the specified context type hierarchy toward more specific value (i.e. obtain value of child context)

Example: Goal to display information on CRT screen may be substituted by a goal to display information on LCD.

Context value substitution

Definition: Obtains an equivalent substitute context value (i.e. sibling value).

Example: Goal to display infromation on Desktop's LCD screen may be substituted by a goal to display information on TabletPC's LCD.

Context type expansion ("context unfolding")

Definition: Obtain values of related context types. This may result in a refined goal, where the plan eventually may overachieve the original goal. This is useful when there are operators with partially satisfied preconditions. Retrieving additional context types may enable selection of operators previously not applicable in the planning process.

Example: Goal requires that information is to be displayed on LCD screen. No LCD screen is detected in the house, and the goal cannot be satisfied. At the same time, a CRT display is detected. By examining available operators the operator display_on_crt where (location in_kitchen) precondition is not applicable as the location of the user is not known. By expanding ContextMesh and unfolding the location context dimension, the user's present location is detected to establish the applicability of information display on the CRT.

Context type contraction ("context folding")

Definition: Removal of the specified context type. Deletes a context goal from the current set of open goals that the planner must achieve.

Example No text to speech service is available. Remove the text-to-speech requirement (i.e. do not take into account activity of the user that implies goal constraint on text to speech service).

Obtain context value at time t
Definition: Obtains a substitute context value at specified point in time.

Example: If a sensor has failed and a context value can not be obtained, a context history is accessed to try and retrieve a past context value. Utility models are used to assign utility values to the context values as well in order to guide the transformation process. For example, a sibling value may be preferable to the higher abstraction.

3) Summary

The contribution of the `ContextMesh` is twofold. Firstly, by placing unified context ontology in the core of the architecture all applications can query it for information relevant to the application's particular domain. More importantly, by providing the ability to measure the success of composition in terms of context preferences (i.e. the utility function), it is the core component of context goal transformation framework that reformulates unreachable goals to problems that can be solved by the planner. It facilitates soft goal satisfaction, allows for scenario-specificity and does not require a single common ontology.

D. Goal Transformation Engine

Goal transformation is applied under two circumstances: firstly when no plan is found for a given goal, and secondly when a plan with a higher utility can be applied to solve a goal, thereby over-achieving the original goal.

The context aware goal transformation algorithm builds on the ideas presented by Cox et al. [16]. They devised a framework for goal transformation in a continuous planner to select appropriate goal transformations automatically. The goal arguments and predicates may be moved along an abstraction hierarchy, enumerated set, a number line, or component containment.

We have extended their algorithm to allow for transformation of context goal conditions as well. The process of goal transformation involves transformations of both core and context goal conditions. We summarize again, the operations provided by `GoalRepository` and `ContextMesh` respectively, which facilitate core and context goal transformation:

Core goal transformations:
1. Goal Repository: Goal type transformations (generalization and specialization)
2. Goal Repository: Goal argument transformation (generalization and specialization)
3. Goal Repository: Goal predicate removal (constraint relaxation)

Context goal transformations:
1. ContextMesh: Context Value Generalization
2. ContextMesh: Context Value Substitution
3. ContextMesh: Context Value Specialization
4. ContextMesh: Context Unfolding
5. ContextMesh: Context Folding
6. ContextMesh: Context Historical Substitution

GoalMorph uses contextual information to drive the transformation process to devise "best-approximation" goals. In addition, we incorporate utility model in the goals, to enable reasoning whether one goal transformation has a higher expected utility than another.

Algorithm 1 outlines the method for context aware goal transformation. GoalMorph firstly it generates a set of pending core and context goal conditions. By applying goal type and goal argument transformation, it attempts to satisfy the core goal - i.e. get the restaurant directions and prepare them for display. If no transformation can be found even to solve the core problem partially, the algorithm stops here.
Once the core goal can be (partially) satisfied, the transformation engine attempts to refine/overachieve it by transforming the context goals as well. For each context type it retrieves current value (or uses it most recent value from `ContextHistory`, if the current value is not available). It then determines the hierarchy of importance for this type in this scenario.

The next step is to compute the set of transformations to be applied on this goal – a search problem, which can be guided by following methods: (1) Domain control knowledge, (2) Utility function and (3) User input.
Domain control knowledge, devised at the domain engineering stage, may include control structures expressing the priorities among goal transformations for the given scenario.

The utility function is to be applied with a learning mechanism, which over the time acquires utilities based on the feedback from the user. The order in which transformations are done is determined in the following manner.

Firstly the utilities of substitute goals are determined to calculate the overall cost-benefit of the substitute solution. Initially we assign synthetic, random utility values to the goal conditions. The next step is to apply learning algorithm to refine these utilities. Upon the goal transformation and execution of the resulting plan, user would provide feedback on the usefulness of the transformation. The feedback is then incorporated and reflected in the utility values of transformations performed. Finally, the user may manually guide the goal transformations. However in this work we are focusing on the automated goal transformation.

The current implementation of this algorithm supports all of the above methods, and they are evaluated in the next section. The order in which transformations are applied and evaluated at present is determined by the utility of the resulting goal - the higher utility the higher priority of the transformation. There may be cases where two transformations have the same utility, the engine in that cases randomly determines which to apply, if it. Otherwise, if there is a case where both goal generalization and specialization are of the same utility, a more refined/specific goal is always

preferred. When conflicting context goal conditions emerge from the transformation the original goal always has precedence over the transformed one, so the context type will be folded and hidden.

Algorithm 1 Context Aware Goal Transformation

```
1: procedure CONTEXTAWAREGOALTRANSFORMATION(S_CG, S_CA)
2:     for all g ∈ S_CG do
3:         transformCoreGoalArgument(g)
4:         if g ≠ solvable then
5:             transformCoreGoalType(g)
6:         end if
7:     end for
8:     for all g_c ∈ S_CA do
9:         h ← keyHierarchy(c)
10:        v ← value(c)
11:        if v ≠ available then
12:            v ← valueAtTime(c, t)
13:            g_c ← generateContextGoal(v)
14:        end if
15:        counter ← 0
16:        T ← getRelevantTransformations(c, v, h)
17:        while g_c ≠ solvable do
18:            for all t ∈ T do
19:                v ← t(c, v, h)
20:                g_c ← generateContextGoal(counter, c, v, h)
21:            end for
22:        end while
23:    end for
24: end procedure

25: procedure GETRELEVANTTRANSFORMATIONS(c, v, h)
26:    Arrayofi ← findApplicableTransformations(c, v, h)
27:    for all i ∈ Arrayofi do
28:        if i = 0 then
29:            v ← contextValueSibling(v, h)
30:            g_c ← generateContextGoal(v)
31:        else if i = 1 then
32:            v ← contextValueChild(v, h)
33:            g_c ← generateContextGoal(v)
34:        else if i = 2 then
35:            v ← contextValueParent(v, h)
36:            g_c ← generateContextGoal(v)
37:        else if i = 3 then
38:            S_CA ← S_CA + unfoldContextType(c)
39:        else if i = 4 then
40:            S_CA ← S_CA − g_c
41:        end if
42:    end for
43: end procedure

44: procedure UNFOLDCONTEXTTYPE(c)
45:    S_CT ← findRelatedContextTypes(c)
46:    O_CT ← orderContextTypesByUtility(S_CT)
47:    for all c ∈ O_CT do
48:        v ← value(c)
49:        g_c ← generateContextGoal(v)
50:        S_CA ← S_CA + g_c
51:    end for
52: end procedure
```

II. EVALUATION

We have implemented the context aware goal transformation framework and conducted initial experiments on our prototype to determine its scalability and impact on the overall performance of the system. The experiments were performed on an 800 MHz, Pentium III with 2 GB RAM.

We modeled our scenario from entertainment domain, described in Section 3, which contained 100 fact and 20 operators. To evaluate the transformation algorithm we randomly removed certain operators from the domain to simulate missing services. ContextMesh contained 7 context dimensions, such as the ones described in Section 4.5, and shown in Figure 10. Each context dimension contained up to 10 different values and their corresponding utilities, which were initially, assigned synthetically random generated values. Furthermore, each context type was associated with one or more hierarchies.

We generated goals containing up to 40 goal literals. Following are some of the sample service requests that were offered by our system:

1. Find a dining or entertainment venue (location-based)
2. Find an entertainment venue (event-based)
3. Find a dining venue (cuisine-based search)
4. Find directions to the venue
5. Book dining or entertainment venue
6. Make booking and find directions for dining and entertainment venue

All the planning is performed by TLPlan [10] in breadth first search mode with no search control knowledge. TLPlan normally uses domain specific search control information to control simple forward chaining search, where the planning operators are applied to the current state to generate its successors. Bacchus et al. [10] demonstrate that that control strategies can be a considerable aid in speeding up the planning (up to 20times) compared to the planning without search knowledge. In this work, however, we focused on the performance of the goal transformation algorithm itself and use the planner without control knowledge. We evaluated our goal transformation algorithm, and compared two different approaches in selection of transformations.

First one applied a random search algorithm, where transformations were selected in a random manner, whilst the second one was utility driven. We ran goal transformation algorithm for each failed goal until the transformed goal (with the highest utility) could be solved, otherwise the transformation was set to terminate after 1sec.

(Partial) Goal Utility and Size To evaluate the utility of transformed goal we employed a formal model to represent partial fulfillment of the overall task, as well as individual goals. We base our approach on the method for expressing the degree of satisfaction of each atomic, atemporal4[4], goal literal, introduced by Haddawy et al.[17]. This allows reasoning about the extent to which the goal is satisfied.
Haddawy et al. further separate atemporal goals into the ones with symbolic and quantitative attributes. For example, a symbolic goal would be `(directions pda_display)`

[4] Atemporal goals describe what needs to be achieved, while temporal ones describe when it is to be achieved. In our work, we focus exclusively on atemporal goals.

and (screen_resolution 1024_768). In our work, we are only concerned with symbolic goals. Haddawy et al. define a degree-of-satisfaction function (DSA) for a symbolic attribute goal, in terms of an application-supplied sequence S of mutually exclusive goal literals *g1, g2 ... g_n*, such that gn is the actual component of the goal and gi represents a greater degree of satisfaction than gj if i < j. DSA is in range [0.0-1.0], where 0.0 is representing no satisfaction and 1.0 is full satisfaction of the goal literal.

Goal ontology from GoalRepository and ContextMesh hierarchy already provide a base for devising a function for core and context goals respectively, to specify partial satisfaction of atemporal goals. At present we assign synthetic values to each goal literal in their corresponding hierarchies, as described in sections 4.3 and 4.5. The overall satisfaction of the whole task is a sum of the partial satisfaction value of each individual goal literal. This can be further extended to incorporate the goal weight to model the priorities for each goal literal. We used this formal model to compare performance of plans generated for goals with and without goal transformation. Furthermore, we ran and compare goal transformation algorithm in random search mode and utility mode for selection of transformations.

We measured the reduction in goal performance in two ways. Firstly, we were interested in the number of goal literals that can be satisfied with and without goal transformation. We compared this to the size of the original goal (i.e. total number of goal literals that are to be satisfied), and number of goal literals that are solved when the planning fails to satisfy completely the original request.
The results of this experiment show that both random based goal transformation (see figure 12) and utility-driven goal transformation find goals that can be solved and are at least 60% of the same size as the original request. The reduction in number of goal literals satisfied is greater in cases without the goal transformation than when planning with such transformations, and as expected the lowest reduction occurs when utility-driven transformations are applied.

However, even the randomized transformation selection algorithm is also an improvement to the planning with the original goal only, as the composition request can be satisfied.
Secondly, we measured the overall goal utility by measuring the sum of the utilities of each goal literal. We compared goal utility with and without goal transformation. Figure 13 shows the utility of the goal after random and utility based goal transformations, when varying the problem complexity from 10 to 40 goals. Reduction in goal utility with goal transformation does not increase with the number of goal conditions introduced. That is due to the fact that there is a higher probability in successfully transforming goals with higher number of goal conditions.
It is important to note, that higher goal size may not imply higher utility. Depending on user preferences shorter goals may incur higher utility.

GoalMorph Latency In this experiment we compared the time that it takes planner to find no solution (i.e. for goal to fail) to the time taken to transform the goal into the one that can be solved and replan, as shown in Figure 14. We compared the total transformation time (taken to generate a goal that can be solved) with the planning time (taken to determine that the goal can not be solved). Whilst, as expected the time difference increases with the number of transformations, for all tested cases transformation time remains sufficiently small (i.e. less then 1600ms for goals of size 40) to justify the overhead introduced by goal transformation.

Context and Core Goal Transformations Performance
We compared transformation time for core and context aware goals and their impact on the overall transformation time. As core and context goals may be dependent, we only compare core and context aware goal transformations that can be performed in isolation (independently). We expected the context aware goal transformation is computationally more complex (up to 8 times) due to the overhead in the communication with the ContextMesh and context layering process, interaction with ContextProxy to generate new context goal(s), as well as interfacing with ContextService to retrieve historical values, as shown in Figure 15. Finally, performance of the context goal transformations also depends
on the size of the size of the context hierarchy.

GoalMorph Scalability Finally we simulated an increase in the goal size and observed systems behavior in terms of the number of transformations generated and the running time of GoalMorph. As shown in Figure 16 the system scales well, being able to generate up to 240 core and context transformations in 0.4seconds in the random transformation selection mode. As expected the utility driven transformation selection mode has a higher running time, due to the search for the transformed goal with highest utility, whereas the random search terminates once a first goal that can be solved is found.

III. RELATED WORK

Most of the previous work in goal oriented service composition [6, 2] has either assumed a static environment where plans are always solvable, or focused on adapting the execution process by replacing service instances. In contrast, our work generates solvable goals even when service replacement is not adequate – for instance, when no services that would satisfy a particular part of a goal are available. In this section we compare our solution to GTrans and discuss previous research in the area of partial goal satisfaction.

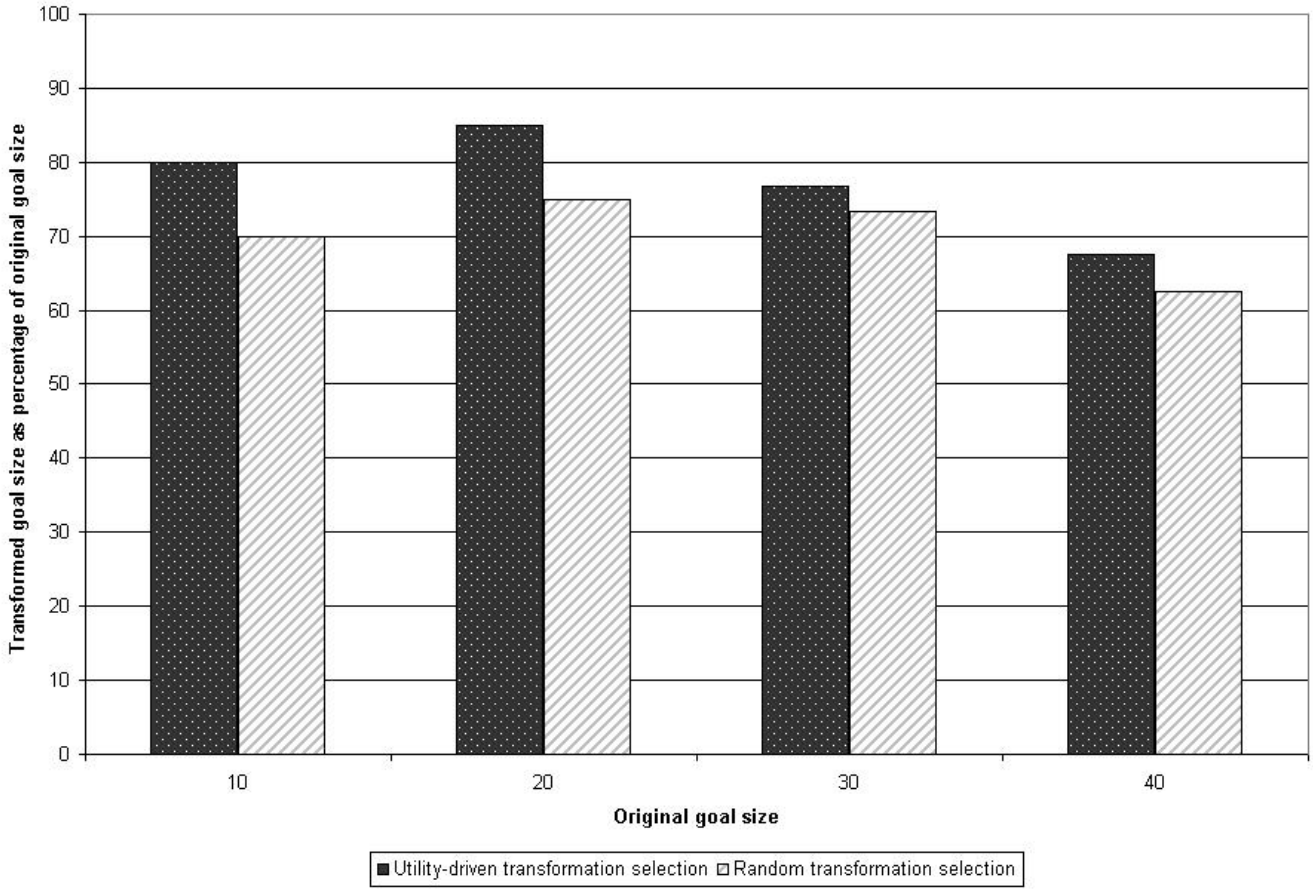


Figure 12 Number of goal solved as a function of goal size using random search and utility transformation selection.

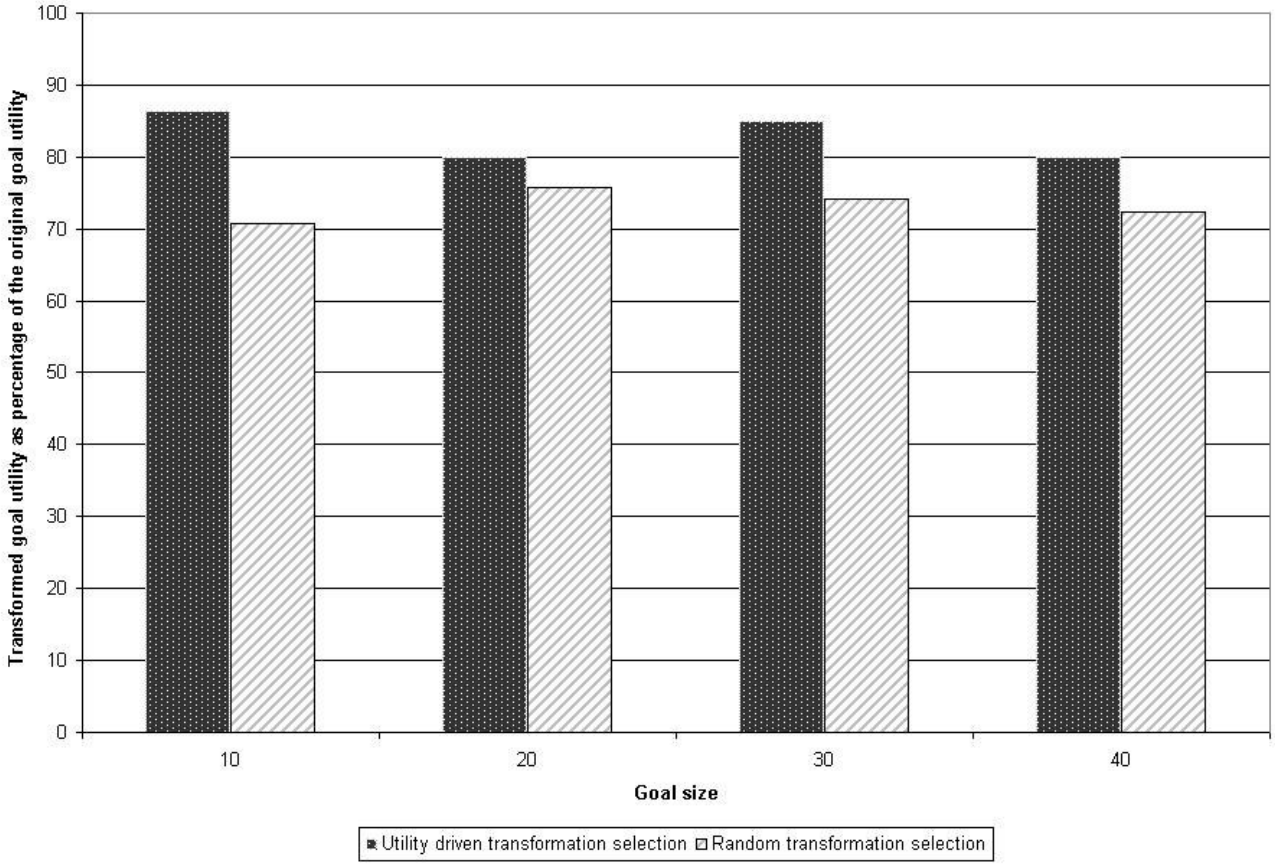


Figure 13 % of achievable utility of transformed goal solved as a function of goal size using random search.

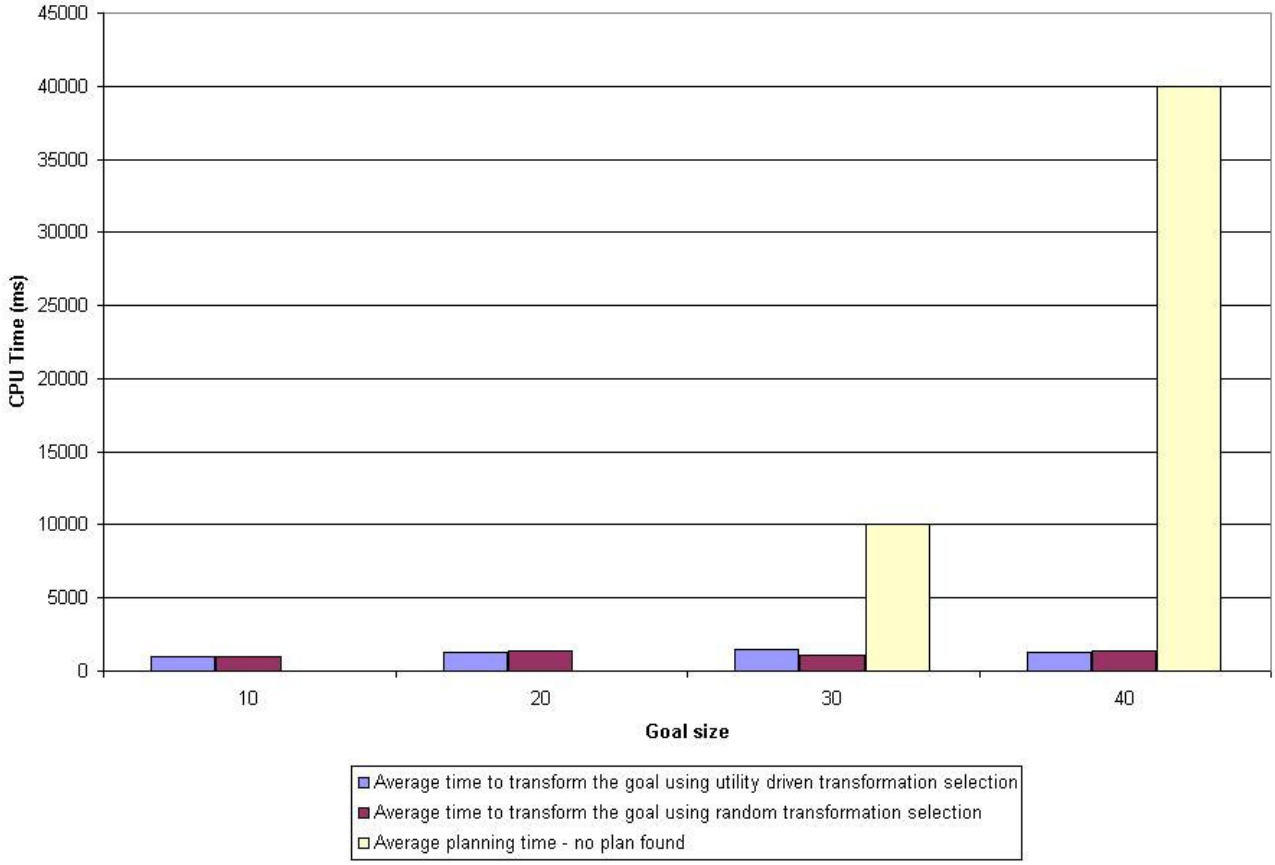


Figure 14 Planning time (no plan found) vs Total transformation time

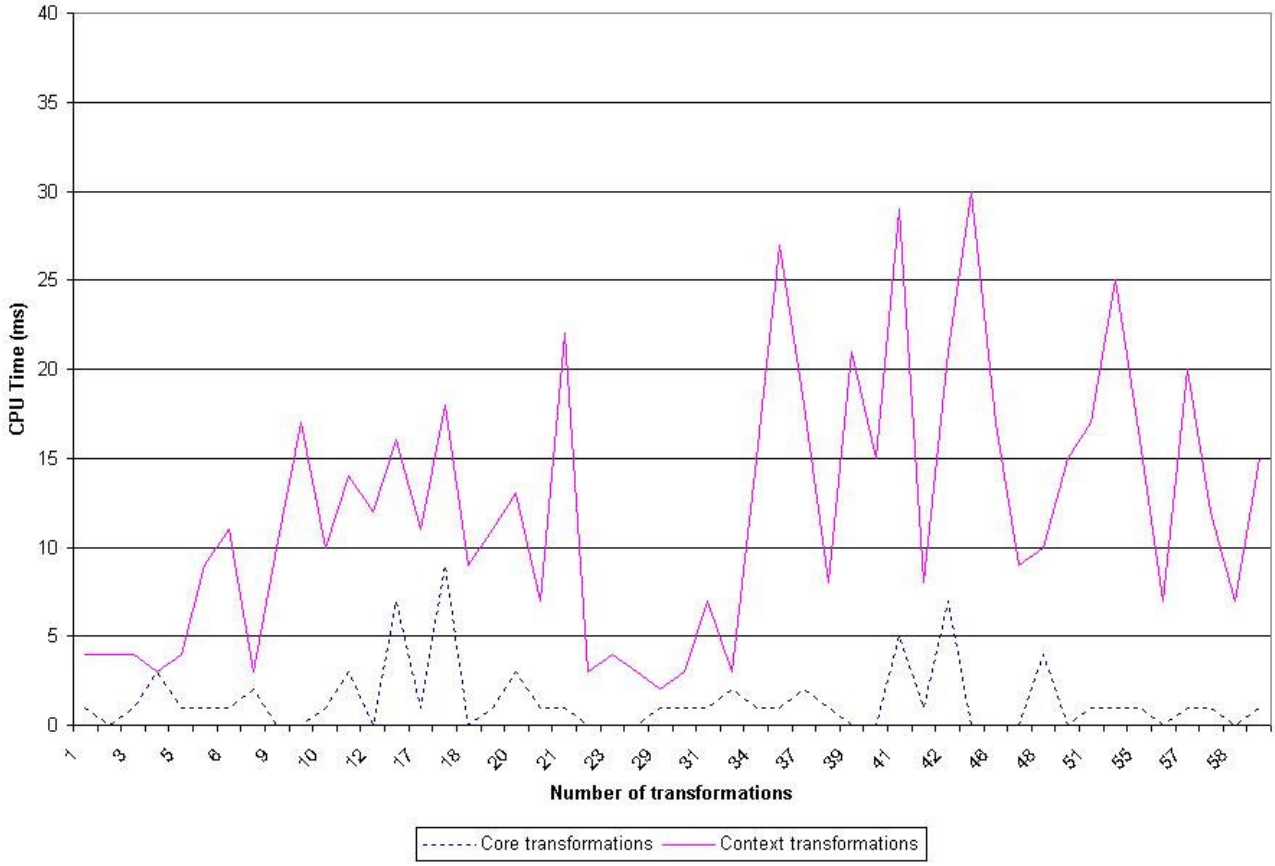


Figure 15 Performance of utility-driven core and context goal transformations as number of transformations increases

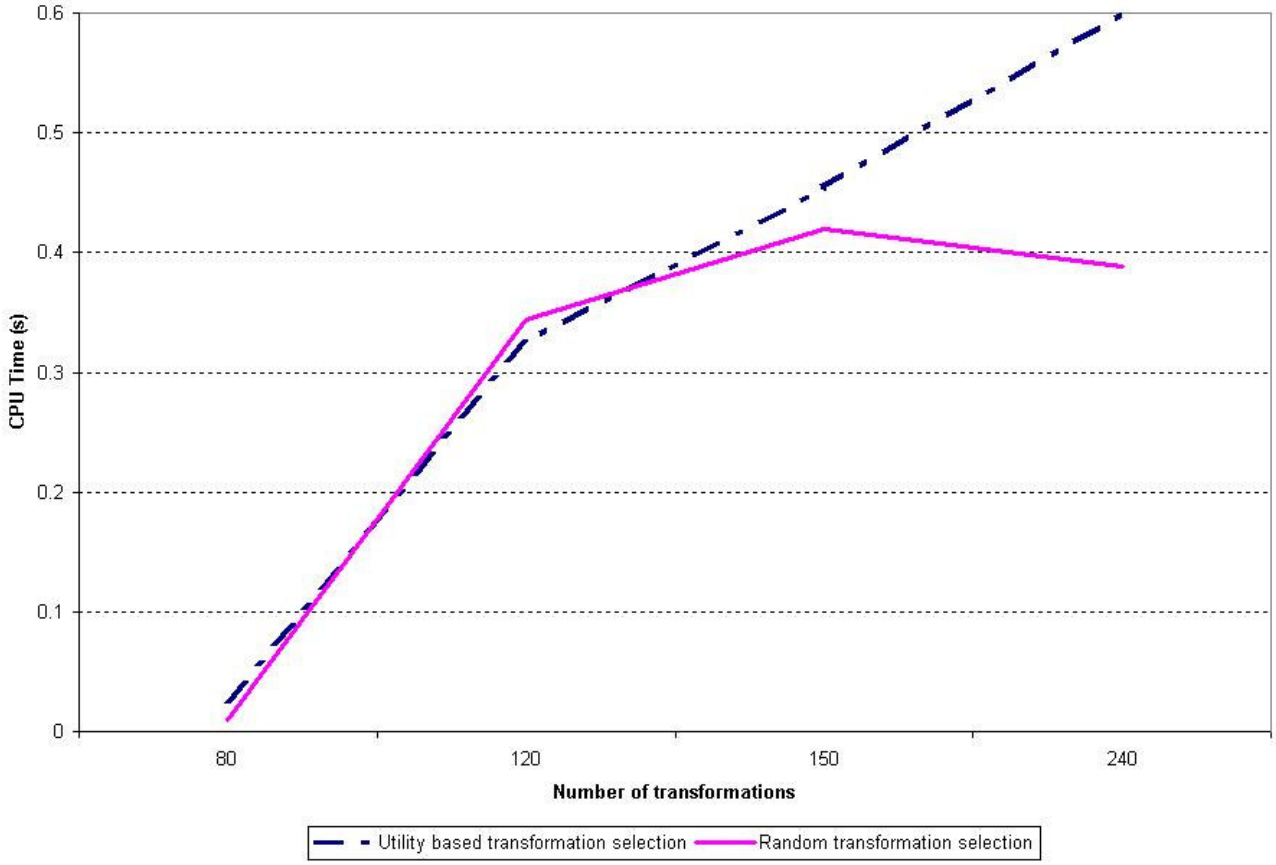


Figure 16 GoalMorph performance as a function of number of transformations.

A. Comparison with GTrans

Cox [16] introduced taxonomy of goal transformations based on an organization of goals and objects in the goal hierarchy, as an approach to planning in a world under continual change. They extend PRODIGY [18] planner to automatically select the appropriate goal transformations in response to world changes to completely solve the transformed problem. Later they apply this approach to mixed-initiative planning [19].

We have extended their ideas by introducing the ContextMesh to enable context aware goal transformation, which allows the system not only to transform the actual goal (e.g. abstracting the goal from finding a Spanish restaurant to any closest restaurant or substituting Spanish with a Mexican), but also to unfold and fold context layers, transform the context goal conditions, to allow for satisfaction
of the most "closest" goal in the "closest" context. Our approach also enables the user to specify importance measure across the context dimensions. We now discuss key differences between our solution and Cox's work.

Selection of Goal Transformations When selecting applicable goal transformations, besides the commonly used unguided search, GTrans relies on domain control knowledge In their later work [19] GTrans supports so called "mixed initiative approach" where users can establish and transform goals manually through visual representation. Our system, in addition to domain control search strategy and unguided search (randomized algorithm), provides a utility model for goals contexts and their corresponding transformations. The utility model is designed to enable support for searching a net benefit solution (i.e. one with a maximum goal utility and minimum cost). The utility based approach makes out approach to the goal transformation a domain independent solution. It is important to mention that we support all three approaches – utility-based, domain based, and manual transformation; the implementation of the prototype employs a hybrid approach to goal transformation. Goals are organized in hierarchies and corresponding transformation are assigned utilities.

User Interaction The user explicitly performs goal transformations in the mixed-initiative version of GTrans.
In contrast, our solution enables the user to provide feedback on the usefulness of a particular solution (after goal transformation) and feeds this into the learning system to update the utilities of corresponding goal transformations.
The design of the GoalMorph allows for system to be extended to include the user directly in the goal transformation loop, however in this paper we focus on automated goal transformation.

Planner Dependence Cox et al. integrate goal transformation with Prodigy, a state space nonlinear planner which follows a means ends analysis backend chaining search procedure that reasons about both multiple goals and multiple alternative operations form its domain theory. Goal Morph on the other hand is planner independent. This is often advantage as we may want to substitute planner with a new one, or may not have access to its source code. Our prototype is implemented around TLPlan. We use internal

representation for goals, their hierarchies and utilities. The system allows for goals to be imported in the PDDL format, and is ontology independent, allowing for any arbitrary goal and context ontology to be incorporated given the necessary translation tools are in place.

Replanning GTrans, is not only planner dependent, but also interleaves goal transformation with the actual plan refinement process. We chose to keep our solution planner independent and therefore it is designed in such a way to rely on re-planning. As re-planning from scratch may be extremely time-consuming, we take the partially created plan from the planner, and incrementally perform goal transformation and replan from there.

Supported Hierarchy Types Dependence GTrans allows for goals (their predicates and arguments) to be organized along the four key types of hierarchies: abstraction hierarchy, number line, enumeration set and patronomy. We extend this to allow for any arbitrary hierarchy to be included.
This is achieved by creating a comparable interface for each hierarchy to define the wanted ordering among hierarchy elements. Optionally, one can import the hierarchy directly.

Goal Priorization Finally, our utility model allows for goals to be prioritized thereby guiding the selection of goal transformations. At this stage of our prototype implementation goals are treated independently.

B. Partial Satisfaction Planning

Goal transformation is related to PSP [17, 20, 21]—an area of research in AI Planning that focuses on ensuring that the resources in the planning problem are not overloaded. All of these projects allow for PSP with goal utilities; however they do not provide mechanism for goal priorization. Briel et al [21] furthermore do not support interacting goals (i.e. goals are assumed to be mutually exclusive). Williamson et al. [20] allow for for partial satisfaction planning with goal utilities, but their system still requires for all the goal conjunctions to be reached.

IV. Discussion and Future Work

To address the goal transformation problem fully, several related challenges need to be overcome. In this section we discuss these and our work in progress in addressing them.

A. Utility Function

Optimizing the match between the needs of the users composite service request and the transformation capabilities corresponds to maximizing user's utility for a specific request.
Our approach is based on the method presented by Poladian et al. [22] for dynamic configuration of resource-aware services to the problem of goal transformation. The user's utility is expressed by means of user preference function that maps from multidimensional transformation space to a uni-dimensional utility space.

Utility is a measure of user happiness with respect to the possible outcomes, or, in other words, a formal representation of how useful a transformed goal is relative to the specific original task. We encode utility in the interval [0,1] of the real numbers, where 0 utility corresponds to the transformed goal being unacceptable for the task, and 1 corresponds to the user satisfaction, in the sense that the increasing it may not improve user's perception of usefulness for the specific task. The transformation capability space is a set of available transformations for each goal literal.

B. Goal Scheduling

Deployment of GoalMorph in the production environment raises questions of scheduling goals that compete for system's resources. Given a number of disjoint goals and their transformations, we need a mechanism to prioritize access to system resources. We envisage a method for prioritization of service requests (and any related goal transformations) by different users to depend on high-level criteria, such as the importance of goals and the subscription class of the users. For instance, users paying the lowest subscription fee may get a lower probability of their goal utility requirements being met, whereas a high subscription user with the same requirements may get a higher probability and a faster service response.

V. Summary

In this paper, we have presented GoalMorph, a framework for context aware goal transformation to facilitate fault tolerant, context aware, service composition, based on the core and context goal transformations. We have introduced ContextMesh, a multidimensional data structure for hierarchical organization of context. It enables context layering – process of controlling the amount of context data used to transform the context goal. By means of an example we demonstrated the use of our algorithm for context aware goal transformation to generate a problem that can be solved by the planner. We demonstrated that the performance of our composition system is practical when context aware goal transformation is introduced. The overhead that GoalMorph introduces is minimal; it performs up to 240 core and context transformations in 0.4seconds whilst randomly selecting transformations. GoalMorph generates partially satisfied goals, which achieve more than 60% of the original utility, despite the increase in the goal size. Future work will incorporate learning methods to refine goal utilities based on user's feedback from generated compositions, and devising a method for prioritization of goal requests.

Acknowledgment

We would like to thank IBM Zurich Research Laboratory for supporting this work and Evangelos Kotsovinos for providing valuable feedback.

References

[1] Zeng, L., Benatallah, B., Lei, H., Ngu, A., Flaxer,D., Chang, H.: Flexible Composition of Enterprise Web Services. Electronic Markets – Web Services 13 (2003)

[2] McIlraith, S., Son, T.: Adapting Golog for Composition of Semantic Web Services. In: Eighth International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France (2002)

[3] Dale, J., Ceccaroni, L.: Pizza and a Movie: A Case Study in AdvancedWeb-Services. (In: AAMAS 2002. Workshop on Challenges in Open Agent Systems)

[4] Ponnekanti, S.R., Fox, A.: SWORD: A Developer Toolkit for Web Service Composition. In: 11th World Wide Web Conference (Web Engineering Track), Honolulu, Hawaii, (2002)

[5] Sabou, M., Richards, D., Splunter, S.v.: An Experience Report on Using DAML-S. In: The Twelfth InternationalWorldWideWeb Conference,Workshop on E-Services and the SemanticWeb (ESSW'03), Budapest, Hungary (2003)

[6] Wu, D., Sirin, E., Hendler, J., Nau, D., Parsia, B.: AutomaticWeb Services Composition Using SHOP2. In: 13th International Conference on Automated Planning & Scheduling. Workshop on Planning for Web Services., Trento, Italy (2003)

[7] Thakkar, S., Knoblock, C.A., Ambite, J.L.: A View Integration Approach to Dynamic Composition of Web Services. In: 13th International Conference on Automated Planning & Scheduling. Workshop on Planning for Web Services., Trento, Italy (2003)

[8] Gelernter, D., Carriero, N.: Coordination Languages and Their Significance. Communications of the ACM 35 (1992) 97–107

[9] Vukovic, M., Robinson, P.: Adaptive, Planning Based, Web Service Composition for Context Awareness. In: Advances in Pervasive Computing. Volume 176. (2004) 247–252

[10] Bacchus, F., Kabanza, F.: Using Temporal Logic to Control Search in a Forward Chaining Planner. In: Proceedings of Second International Workshop on Temporal Repre sentation and Reasoning (TIME), Melbourne Beach, Florida (1995)

[11] Vukovic, M., Robinson, P.: GoalMorph: Partial Goal Satisfcation for Flexible Service Composite. In: Next Generation Web Services Practices (NWeSP). (2005)

[12] Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, NJ (1995)

[13] Lei, H., Sow, D.M., Davis, II, J.S., Banavar, G., Ebling, M.R.: The design and applications of a context service. SIGMOBILE Mob. Comput. Commun. Rev. 6 (2002) 45–55

[14] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M.,Weld, D.,Wilkins, D.: PDDL— The Planning Domain Definition Language (1998)

[15] Fikes, R.E., Nilsson., N.J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artifical Intelligence 2 (1971) 189–208

[16] Michael T. Cox and Manuela M. Veloso: Goal Transformations in Continuous Planning. In: Proceedings of the AAAI Fall Symposium on Distributed Continual Planning, AAAI Press (1998)

[17] Haddawy, P., Hanks, S.: Utility Models for Goal-Directed Decision Theoretic Planners. Technical Report TR-93-06-04 (1993)

[18] Veloso, M., Carbonell, J., P'erez, M., Borrajo, D., Fink, E., Blythe, J.: Integrating planning and learning:The prodigy architecture. Journal of Experimental and Theoretical Artificial Intelligence 7 (1995) 81–120

[19] Cox, M.T., Zhang, C.: Planning as a mixed-initiativegoal manipulation process. Technical Report WSUCS-04-02, Wright State University, Department of Computer Science and Engineering, Dayton, Ohio (2004)

[20] Williamson, M.: Optimal planning with a goaldirected utility model. In Hammond, K.J., ed.: Proceedingsof the Second International Conference on AI Planning Systems, Menlo Park, California, American Association for Artificial Intelligence (1994) 176– 181

[21] van den Briel, M., Nigenda, R.S., Do, M.B., Kambhampati, S.: Effective approaches for partial satisfaction (over-subscription) planning. In McGuinness,D.L., Ferguson, G., eds.: AAAI, AAAI Press / The MIT Press (2004) 562–569

[22] Poladian, V., Sousa, J.P., Garlan, D., Shaw, M.: Dynamic configuration of resource-aware services. In:ICSE '04: Proceedings of the 26th International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2004) 604–613

Author Biographies

Maja Vukovic is a PhD candidate at Computer Laboratory at University of Cambridge. Her research is supported by IBM Zurich Laboratory. Maja Vukovic obtained her B.Sc. in computer science and mathematics from the University of Auckland in New Zealand. She received an M.Sc. degree in computer science from the International University in Bruchsal, Germany for her work on in-vehicle location-based systems.

Previously she worked as a Research Scientist at DaimlerChrysler Research Lab in Palo Alto, CA, where she investigated condition monitoring of vehicles and telediagnostics, as well as open telematic platforms. Her interests are in the area of context aware computing and service oriented architectures.

Peter Robinson is Professor of Computer Technology and Deputy Head of the Computer Laboratory, University of Cambridge, working in the Rainbow Group on computer graphics and interaction.

He is a Fellow, Praelector and Director of Studies in Computer Science at Gonville and Caius College. He is also a Chartered Engineer and a Fellow of the British Computer Society.

Professor Robinson has been leading work on the use of video and paper as part of the user interface. Recent work has included desk-size projected displays and inference of users' mental states from video images of their faces.