Adaptation Space: A Design Framework for Adaptive Web Services

Henrique Paques, Ling Liu, and Calton Pu

Georgia Institute of Technology
College of Computing
Atlanta, GA 30332-0280 USA
{paques,lingliu,calton}@cc.gatech.edu

Abstract. Web service adaptation is an important feature for mission critical Web services. It is widely recognized that thrashing and crashes occur in system saturation for many statically adaptive resource management algorithms, including CPU, memory, and network congestion. Service adaptation supports alternative responses to saturation control, maintaining service and server system stability and progress, instead of thrashing. In this paper, we present a design framework for developing adaptive web services. The core of this framework is the adaptation space model, which is based on the concepts of adaptation space and adaptation case. An adaptation space is defined by a use context and a partial order of adaptation cases. Each adaptation case describes a specific adaptation of a program or component of a web service. There are three main thrusts of the adaptation space approach. First, it defines a multi-dimensional adaptation context for capturing and coordinating different kinds of adaptation at different levels of a web service. Second, it provides a uniform way for representing and viewing a collection of alternative adaptations for a given web service component. Third, it promotes a declarative and incremental approach to adaptation specification, allowing the incorporation of new adaptation behavior of a web service in terms of existing adaptation cases. We evaluate the adaptation space approach using Ginga, an adaptive query processing service for handling queries over multiple data sources with diverse capabilities across the Internet. Our experimental results show that Ginga query adaptation can achieve significant performance improvements (up to 40% of response time gain) for processing distributed queries over the Internet in the presence of end-to-end delays.

Keywords: program adaptation, adaptation space.

1 Introduction

Most of the current efforts on developing web services concentrate mainly on how to guarantee that these services can interoperate. Well-known examples of these

M. Jeckle and L.-J. Zhang (Eds.): ICWS-Europe 2003, LNCS 2853, pp. 49–63, 2003.
© Springer-Verlag Berlin Heidelberg 2003

efforts are the WSDL[1], SOAP[2], and UDDI[3] standards. However, an important problem that still needs to be solved is related to the design of web services. Assuming that these services are executed in a distributed open environment that dynamically changes over time, how can we guarantee the maintenance of the performance optimality of these services at runtime. To illustrate this problem, take a web service S that needs to contact other remotely located services in order to complete its task. Considering that the implementations of web services are independent from each other, and that the bandwidth of the network connections between these services can fluctuate quite unpredictably, it is difficult to anticipate the performance behavior of each service. Consequently, it is very likely that S will have unpredictable performance. The problem becomes aggravated when S is responsible to execute a task that is part of a mission critical application, where each task has a specific deadline to be completed.

In order to overcome unpredictable changes to the runtime environment, such as network bandwidth fluctuations, the web service needs to be able to *adapt* its program (or component) execution accordingly. If we take service S, one possible way to adapt its execution when a remote web service W is taking too long to respond to a S's request, is to contact an alternative service that can provide similar response to the same request. Another possible adaptation of S for coping with the same delay problem is to modify S's program (if possible) to perform other parts of its task while waiting on W's response. A third adaptation is to close and re-open the connection with W and submit the request again. It is possible that the previous route established between S and W was too crowed, resulting in network congestion. Observe that we can have more than one alternative for adapting a given web service. The challenge is then how to coordinate and integrate these alternative adaptations.

In this paper we propose an adaptation space based approach to web service adaptation. The core of this adaptation model is the notion of adaptation space and the mechanisms for coordinating and integrating different kinds of adaptations. An adaptation space is defined by a use context and a partial order of adaptation cases. Each adaptation case describes a specific adaptation of a program or component of a web service. Given a program or component, say T, the adaptation process typically consists of three steps:

- **Step 1: What to Adapt.** Construct an adaptation space for T where preferred adaptation alternatives are specified by a use context and a partial order of adaptation cases (see Sect. 3 for concrete syntax);
- **Step 2: When to Adapt.** Monitor the use conditions of each specified adaptation alternative in the adaptation space of T, and notify the adaptation engine whenever a use condition becomes true;
- **Step 3: How to Adapt.** Turn on the appropriate adaptation case to prevent the running program or component from unnecessary lost of performance, thrashing, abnormal exit, or crash. The selected adaptation case fulfills this

[1] `http://www.w3.org/TR/wsdl`
[2] `http://www.w3.org/TR/SOAP`
[3] `http://www.uddi.org`

job by redirecting the ongoing program to run using specialized code that can accomplish the intended task through partial evaluation, such as using specialized code with either fewer resources or requiring fewer capabilities.

There are three main thrusts of adaptation space based approach: First, program adaptation presents a unified framework for representing and viewing of multi-dimensional adaptation context and adaptation opportunities for a given program or component of a web service. Second, program adaptation provides a uniformed way of capturing and coordinating different kinds of adaptation at different levels of a web service. Third, program adaptation promotes a declarative and incremental approach to define the multi-dimensional adaptation context and adaptation alternatives, allowing the seamless incorporation of new adaptation behavior of a web service into the existing adaptation framework.

The rest of this paper proceeds as follows: Sect. 2 motivates the adaptation space based approach using a real world example. Sect. 3 provides an informal introduction to the concept, the syntax, and the semantics of adaptation cases and adaptation spaces. Sect. 4 presents the formal semantics of the adaptation space model. We illustrate the adaptation space concept and semantics by walking through the application scenario introduced in Sect. 2. Sect. 5 examines the issues that should be addressed by the adaptation space model. Sect. 6 describes a concrete application for the proposed adaptation space model. We discuss related work in Sect. 7 and conclude the paper in Sect. 8.

2 Motivating Application Scenario

Let us briefly motivate the adaptation space based approach using an application scenario drawn from the Bioinformatic domain: One of the common applications of the growing number of Bioinformatic data sources[4] is the discovery of new drugs that may efficiently treat serious diseases caused by virus such as Human Immunodeficiency Virus (HIV). As we describe next, such applications will involve contacting different web services that have unpredictable response times. The main challenge in this scenario is to execute a user request over these unstable remote services with expected responsiveness.

Example 1 (Application Scenario). Consider Next Generation Drugs (NGD), a company specialized in developing state-of-the-art drugs. For each new drug that NGD starts to develop, the first step is to gather all the needed information about the virus that the new drug will attack. In general, this information describes the nucleotides, proteins, and protein structures associated with the virus. Once the necessary information has been collected, the pharmacologist can then start the process of finding the new drug.

Let us assume that NGD has an internal web service NGD-WS responsible for gathering the needed data for starting the development of a new drug. The pharmacologist enters the virus name, and NGD-WS executes query Q expressed

[4] There are more than 500 Bioinformatic data sources available on the Internet.

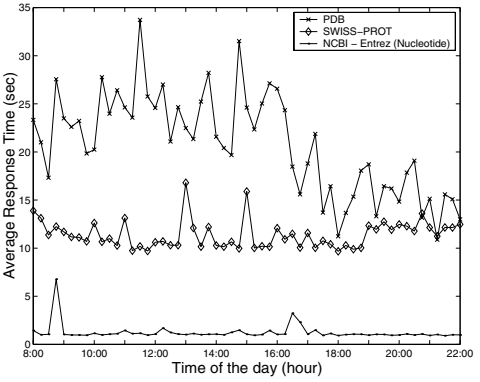


Fig. 1. Average response time measured every 15 min. for 32 days (from 4/11/02 to 5/13/02) between 8:00 and 22:00 EST. Keyword search on "HIV" using PDB, SWISS-PROT, and NCBI Entrez.

in a SQL statement[5] shown in Fig. 2(a), and returns the result back to the pharmacologist. The information on nucleotides, protein, and protein structures is typically provided by remote web services such as NCBI Entrez[6](nucleotides), SWISS-PROT[7](proteins), and PDB[8] (protein structures). However, the response time from these web services can be very unpredictable as shown in Fig. 1. Even though NCBI Entrez has fast response time (in the order of a few seconds), waiting for PDB results can take up to 34 seconds depending on what time of the day the query request was posted to the site. Considering that NGD-WS has to combine this data collected before delivering them to the pharmacologist, such difference in response time could become costly. The situation may become aggravated when NGD needs to execute this type of queries repeatedly (practically, one for each new chemical compound that can potentially affect the virus). The accumulative effect of these delays can significantly affect the company's productivity.

One possible solution for NGD-WS is to adapt the execution of query Q by modifying the data collection and integration process so that the fluctuation on end-to-end delays can be masked. We illustrate next how to implement this solution using adaptation space.

Example 2 (Adapting to End-to-End Delays). Let us assume that NGD-WS generates the optimized query plan P_0 (see Fig. 2(b)) for executing Q, and that the expected transfer rate for the network connection[9] L_i is $Rate(L_i) = 1Mbps$, for $1 \leq i \leq 3$. Now, suppose that as NGD-WS starts the execution of P_0

[5] Any other query language, such as XQuery, could very well be used instead.

[6] `http://www.ncbi.nlm.nih.gov/Entrez`

[7] `http://ca.expasy.org/sprot`

[8] `http://www.rcsb.org/pdb`

[9] We use the term *end-to-end delay* to refer to the delays experienced by the clients due to both bandwidth fluctuation in networks and contention at the servers. In this paper, we concentrate our study on end-to-end delays caused by bandwidth

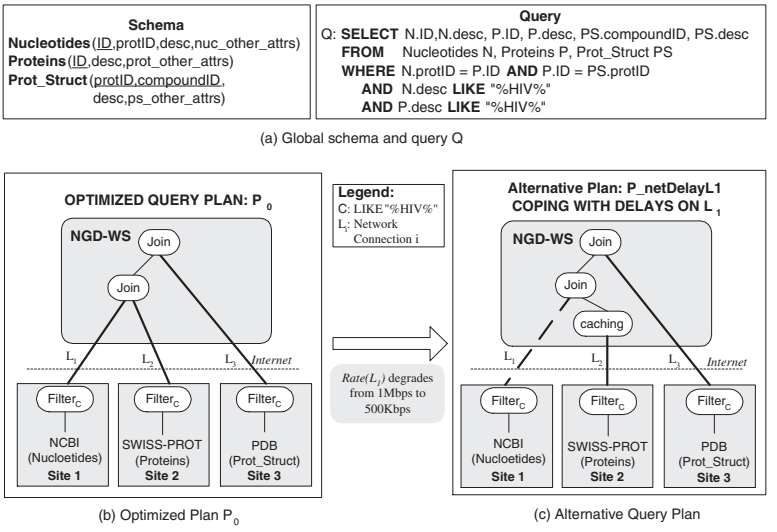


Fig. 2. (a) Query Q and associated global schema; (b) Optimized query plan P_0; (c) Alternative query plan $P_{netDelayL1}$ for coping with delays on network connection L_1.

the transfer rate of connection L_1 degrades from 1Mbps to 500Kbps causing the data from *NCBI* to be slowly delivered. Assuming that P_0 is executed according to the iterator-based model [3], i.e., query operators are scheduled in a left-deep recursive way, without adaptation, the execution of P_0 will be delayed. One way to solve this problem is to start collecting the data from *SWISS-PROT* and caching it locally *while* slowly receiving data from *NCBI*. Fig. 3 presents a sketch of such an adaptation case, declaring an adaptation opportunity: mask the slow delivery of data from *NCBI* while executing other parts of the query. If *NCBI* is completed downloaded by the time the data from *SWISS-PROT* is fully cached, the query execution can then proceed without affecting considerably the final response time.

Based on this adaptation specification, the adaptation engine can infer that this adaptation can be done at compile time if the connection to remote source *NCBI* is known to be slow. Otherwise, the adaptation must be done at runtime, and query plan P_0 will be replaced by $P_{netDelayL1}$ shown in Fig. 2(c).

3 Adaptation Space Model: An Overview

3.1 Adaptation Cases

Adaptation cases define the adaptation behavior of a web service program in anticipation of resource shortage or in the need for performance optimization. More concretely, an adaptation case defines a context for adaptation through program

fluctuation, which results in slow delivery, i.e., the data from the remote server arrives at a rate slower than expected.

ADAPTATION CASE ac_caseNetDelayL1 **adapts execution of query** Q.
goal: query adaptation when Rate(L_1) degrades.
context
resource objects
$Nucleotides$ = {NCBI}; $Proteins$ = {SWISS-PROT}; $Prot_Struct$ = {PDB};
$NetworkLinks = \{L_1, L_2, L_3\}$
resource object parameters
must-provide
ID, desc **from** $Nucleotides$; ID, desc **from** $Proteins$;
compoundID, desc **from** $Prot_Struct$; bandwidth **from** $NetworkLinks$
require
$Nucleotides$.protID, $Proteins$.ID **where** $Nucleotides$.protID = $Proteins$.ID
$Proteins$.ID, $Prot_Struct$.protID **where** $Proteins$.ID = $Prot_Struct$.protID
$Nucleotides$.desc **where** $Nucleotides$.desc LIKE $virusName$
$Proteins$.desc **where** $Proteins$.desc LIKE $virusName$
$Prot_Struct$.desc **where** $Prot_Struct$.desc LIKE $virusName$
adapted query plan: $P_{netDelayL1}$ (see Fig. 2(c))
use-condition
$(Rate(L_1) < 1Mbps) \wedge (Rate(L_2) \geq 1Mbps) \wedge (Rate(L_3) \geq 1Mbps) \wedge$
available($Nucleotides$) = `true` $\wedge$ available($Proteins$) = `true` $\wedge$ available($Prot_Struct$) = `true`
adaptation action
"collect and cache data from $Protein$ while slowly receiving data from $Nucleotide$."
end ac_caseNetDelayL1

Fig. 3. Adaptation case that adapts the execution of Q when $Rate(L_1) < 1Mbps$.

ADAPTATION CASE ac_initialP **adapts execution of query** Q.
goal: fast response time with survivability to network delays.
context
resource objects
// Same as **resource objects** described in ac_caseNetDelayL1 (Fig. 3).
resource object parameters
// Same as **resource object parameters** described in ac_caseNetDelayL1 (Fig. 3).
adapted query plan: P_0 (Fig. 2(b))
use-condition
available($Nucleotides$) = `true` $\wedge$ available($Proteins$) = `true` $\wedge$ available($Prot_Struct$) = `true` $\wedge$
$\{\forall \ell \in NetworkLinks, Rate(\ell) \geq 1Mbps\}$
adaptation action
none.
end ac_initialP

Fig. 4. Initial adaptation case definition.

specialization or generalization. For instance, it identifies what components of a web service should be specialized for adaptation to resource availability or for performance optimization in the presence of certain quasi-invariants [2,13,11].

Generally, adaptation cases are concerned with data and control flow exchanges between a collection of program components, modules, and systems. Resources[10] are the currency of exchange among these system components. In object-oriented systems, all resources are ultimately provided by *classes*. However, it is not always the case in most of the legacy systems. Therefore, in the proposed adaptation space model, both classes and modules are units that provide a set of resources and require another set of resources.

[10] A resource is any entity that can be named in a programming language (e.g., variables, type definitions, etc.) and which can be made available for reference by other program components within a large software system.

ADAPTATION CASE ac_netDelayL2 **adapts case** ac_initialP.
goal: query adaptation when Rate(L_2) degrades.
context
resource objects
// Same as **resource objects** described in ac_caseNetDelayL1 (Fig. 3).
resource object parameters
// Same as **resource object parameters** described in ac_caseNetDelayL1 (Fig. 3).
adapted query plan: P_{12} (see Fig. 8)
use-condition
$(Rate(L_2) < 1Mbps) \wedge (Rate(L_1) \geq 1Mbps) \wedge (Rate(L_3) \geq 1Mbps) \wedge$
available($Nucleotides$) = `true` $\wedge$ available($Proteins$) = `true` $\wedge$ available($Prot_Struct$) = `true`
adaptation action
"collect and cache data from *Prot_Struct* while slowly receiving data from *Protein*."
end ac_netDelayL2

Fig. 5. Adaptation case that adapts the execution of Q when $Rate(L_2) < 1Mbps$.

ADAPTATION CASE ac_netDelayL12 **adapts case** ac_netDelayL1.
goal: query adaptation when Rate(L_1) and Rate(L_2) degrade.
context
resource objects
// Same as **resource objects** described in ac_caseNetDelayL1 (Fig. 3).
resource object parameters
// Same as **resource object parameters** described in ac_caseNetDelayL1 (Fig. 3).
adapted query plan: P_{21} (see Fig. 8)
use-condition
$(Rate(L_1) < 1Mbps) \wedge (Rate(L_2) < 1Mbps) \wedge (Rate(L_3) \geq 1Mbps) \wedge$
available($Nucleotides$) = `true` $\wedge$ available($Proteins$) = `true` $\wedge$ available($Prot_Struct$) = `true`
adaptation action
"collect and cache data from *Prot_Struct* while slowly receiving data from *Nucleotides* and *Proteins*."
end ac_netDelayL12

Fig. 6. Adaptation case that adapts the execution of Q when $Rate(L_2) < 1Mbps$ *and* $Rate(L_2) < 1Mbps$.

Specifically, given a target web service and a reference program unit, an adaptation case description includes

- *Case Name*, described by a character string;
- *Goal Description* (optional), specified by the synopsis of the goal;
- *Context* of the case, defined by (1) a list of resource parameters internal to the reference program unit and critical to the specification of adaptation behavior of the reference program; and (2) a pointer to the adapted code for the reference program;
- *Use Condition*, expressed by a Boolean expression that indicates when it is appropriate to use this adaptation case. Variables in use conditions are defined in the adaptation case context;
- *Adaptation Action*, described by an action to be fired when the use condition is satisfied.

To enable on-demand generation of adaptation action, we need to capture type, state, and alternation of each resource parameter specified in the adaptation case context. The **type** of a resource parameter can be either primitive types such as *integer, char, float, Boolean*, or structured types as those defined

in object-oriented systems. The **state** of a resource parameter includes *provide, must-provide, require, must-require*. Such state information distinguishes (1) the *required* resources from the *provided* resources, and (2) the *mandatory* provide-parameters or require-parameters from *optional* ones. The attribute **alternation** indicates the possible alternatives of the given resource and a list of degradation attributes such as extra-cost, freshness, extra-overhead, to name a few.

Example 3. The initial adaptation case for the execution of query Q is depicted in Fig. 4. It describes the target program (in this case the query Q) in terms of the adaptation model syntax and the particular adaptation context defined in its adaptation space (see Sect. 3.2). For instance, from the SQL statement given in Fig. 2(a), Q provides two output data items from each input source (i.e., *Nucleotides*, *Proteins*, and *Prot_Struct*), and requires the input condition that the data collected is relevant to HIV. We list these resources along with the network links in the initial case ac_initialP as critical parameters for defining the adaptation behavior of Q.

We can define as many as needed adaptation cases for a given query execution plan. For instance, in addition to degradation of $Rate(L_1)$ (Fig. 3), we can also define one adaptation case for Q that deals with degradation of only $Rate(L_2)$ (Fig. 5) and another that deals with degradation of both $Rate(L_1)$ and $Rate(L_2)$ (Fig. 6).

The automatic generation of program adaptation code from the original code may not always be feasible, especially when the target software is a legacy system with sophisticated procedures and state transitions. The development of mechanisms for semi-automatic or automatic generation of specialized adaptation code for given adaptation cases are part of our ongoing research work.

3.2 Adaptation Space

An adaptation space consists of a reference context, which sets up the scope of adaptation behavior and a partial order of adaptation cases, each case representing a specific way of adapting the same program unit. In addition to the set of adaptation cases, an adaptation space description includes the following header information:

- *Name* of the adaptation space, served as a unique identifier;
- *Goal Description*, consisting of a list of objectives and a set of quasi-invariants for this adaptation space;
- *Reference Context*, describing the common resources that all adaptation cases may share, and the nominal functionality and behavior of the program component that this adaptation space deals with. A reference context can be an initial adaptation case, which describes the original, unspecialized program component. The reference context has three main components: (1) the initial case (also called the root case); (2) the set of common resource parameters that all adaptation cases may share; and (3) the set of local parameters used in the goal description and use condition specification.
- *Use Condition*, specifying the general condition to turn on this adapt. space.

Example 4. The initial adaptation case ac_initialP (Fig. 4) and the specialized adaptation cases for coping with network delays (Fig. 3, Fig. 5, and Fig. 6) together form an adaptation space named as_spaceQ as shown in Fig. 7. The reference context of as_spaceQ includes the resource objects and their descriptive attributes that are involved in Q, in addition to the initial adaptation case ac_initiaP. This adaptation space is equipped with a single goal: to cope with network delays with the price of increasing memory contention once concurrent caching processes are fired.

ADAPTATION SPACE as_spaceQ **adapts the execution of query** Q.
  goal: g_1: fast response time with survivability in the presence of network delays.
  reference context
    initial-case: ac_initialP;
    resource-object: *Nucleotides*
      parameters
        ID, protID, desc, nuc_other_attrs; **instance:** *NCBI*
    resource-object: *Proteins*
      parameters
        ID, desc, prot_other_attrs; **instance:** *SWISS-PROT*
    resource-object: *Prot_Struct*
      parameters
        prot_ID, compound_ID, desc, ps_other_attrs; **instance:** *PDB*
    resource-object: *NetworkLinks*
      parameters
        bandwidth; **instances:** L_1, L_2, L_3
  use-condition
    $\exists x \in NetworkLinks, Rate(x) < 1Mbps$
end as_spaceQ

Fig. 7. Example of an adaptation space description for the execution of query Q.

4 Adaptation Space Model: Formal Semantics

In this section we formally describe the concept of adaptation case, adaptation space, and related notions (such as adaptation-of relationship) using graph-based notation. The formal definitions introduced in this section will be served as THE basis for defining adaptive coordination mechanisms that guarantee the correct use of adaptation cases and adaptation spaces. Due to space limitation, in this section we only formalize the key concepts of the adaptation space model. A more detailed discussion on this model can be found in [10].

Given a program component T in a target web service, we can define the adaptation behavior of T through introduction of multiple adaptation cases. Each adaptation case is applicable when its use-conditions become true.

Definition 1 (Use Condition). Let T denote a program component in a target web service. Let $Param(T)$ denote the list of adaptation context parameters of T, and $dom(P_i)$ denote the domain of possible values of an adaptation parameter P_i ($P_i \in Param(T)$). A use-condition, denoted by F, can be defined recursively as follows:

(a) $F = \emptyset$ is a use-condition (empty condition);
(b) $F = P_i \Theta P_k$, $P_i \Theta v_{ij}$, or $v_{ij} \Theta P_i$ are use-conditions, where
  (i) P_i and P_k are comparable functions returning atomic values, $P_i, P_k \in Param(T)$, $v_{ij} \in dom(P_i)$, and $\Theta \in \{==, =, \neq, <, \leq, >, \geq\}$;
  (ii) P_i and P_k are comparable functions returning set values, $v_{ij} \subseteq dom(P_i)$, and $\Theta \in \{==, =, \neq, \subset, \subseteq\}$;
(c) If F, F_1, and F_2 are use-conditions, then the conjunction $F_1 \wedge F_2$, the disjunction $F_1 \vee F_2$, and the negation $\neg F$ are use-conditions;
(d) If F is a use-condition, and x is a free variable in F, then $(\exists x)F(x)$ and $(\forall x)F(x)$ are use-conditions;
(e) Nothing else is a use-condition. □

Definition 2 (Adaptation Case). Let T denote a program component in a target web service. An adaptation case of T is described by a tuple $(N, TARG, GOAL, CTX, UC, AACTION)$ where

- N is the name of the adaptation case.
- $TARG$ is the identifier of the target adaptation unit. It can be a regular class, a module, a program, a procedure, or an existing adaptation case.
- $GOAL$ describes the goal description of the adaptation case.
- CTX is the context description of the case, defined by three components: a pointer to the adapted code, the *list of resource objects* used in the reference program, and the *list of resource object parameters*, where each parameter is further described by a quadruple $(Pr, State, Type, Rdom)$, where Pr is a name for the resource parameter; $State(Pr)$ specifies one of the following use states of the resource parameter Pr: *provide*, *must-provide*, *require*, *must-require*; $Type(Pr)$ defines the type of the resource parameter Pr and optionally the domain constraint of the parameter Pr; and $Rdom(Pr)$ defines the set of permissible values of the resource parameter Pr.
- UC is a set of use-condition, expressed by a Boolean expression, indicating when it is appropriate to use this adaptation case. Variables in use conditions are defined in the adaptation case context.
- $AACTION$ is the adaptation action to be fired when UC is satisfied. □

Example 5. Recall adaptation case ac_netDelay2 (Fig. 5). According to Definition 2, we can describe this adaptation case as follows:

N(ac_netDelayL2) = ac_netDelayL2.
$TARG$(ac_netDelayL2) = ac_initialP.
$GOAL$(ac_netDelayL2) = "query adaptation when Rate(L_2) degrades".
CTX(ac_netDelayL2) = { (P_{12}), (*NetworkLinks*, {(*bandwidth*, must-provide)}),
  (*Proteins*, { (*ID, desc*, must-provide)}), (*Nucleotides*, {(*ID, desc*, must-provide)})
  (*Prot_Struct*, { (*compoundID, desc*, must-provide)})}
UC(ac_netDelayL2) = {($Rate(L_2) < 1Mbps$)∧ ($Rate(L_1) \geq 1Mbps$) ∧ ($Rate(L_3) \geq 1Mbps$)∧
  available(*Nucleotides*) = `true` ∧ available(*Proteins*) = `true` ∧ available(*Prot_Struct*) = `true` }
$AACTION$(ac_netDelayL2) = "collect and cache data from *Prot_Struct* while slowly receiving data from *Protein*."

Definition 3 (Adaptation-of Relationship). Let T denote a program component in a target web service. Let α and β denote two adaptation cases of T, where each case is described by a tuple $(N, TARG, GOAL, CTX, UC, AACTION)$. The case β is said to have an *adaptation-of* relationship with the case α, denoted by $\alpha \Longrightarrow \beta$, if and only if the following conditions are verified:

1. $TARG(\beta) =$ case α.
2. $\forall x \in CTX(\beta) \exists y \in CTX(\alpha)$ s.t. $identical(x, y) \vee$ $(specialized_version_of(x, y) \wedge N(x) = N(y))$.
3. $\forall p \in UC(\alpha) \exists q \in UC(\beta)$ s.t. $degradation_of(q, p)$.
4. $\forall p \in UC(\alpha) \forall q \in UC(\beta)$, $(q = p) \vee degradation_of(q, p)$.
5. $AACTION(\beta) = AACTION(\alpha) + new_action$ □

This definition states that new adaptation cases can be defined incrementally in terms of existing adaptation cases, rather than being defined from scratch. However, the intuition between supercase and subcase is quite different from the superclass and subclass relationship in object-oriented systems. For example, a subcase in the adaptation model may have more restricted use conditions but may consume fewer resources than its supercase. More concretely,

- Condition (1) specifies that each subcase takes its immediate supercase as the target unit of adaptation.
- Condition (2) amounts to saying two points: (i) not every resource of a supercase will be used in its subcases, and (ii) for any resource of a subcase, its implementation may be more specialized than the corresponding one in its supercase (e.g., an alternative remote data source, which has better response time, can be accessed instead).
- Condition (3) states that there must exist at least one condition in $UC(\beta)$ that represents a *degradation of* some resource parameter used by T.
- Condition (4) asserts that the state of all resource parameters listed in case α are either degraded or remain unchanged in case β.
- Condition (5) specifies that the adaptation action of a subcase consists of all the actions considered by its supercase plus a new one.

In general, given an adaptation case, each of its immediate subcases describes an adaptation alternative. The subcases that are siblings with one another may have use-conditions that are either mutually exclusive or overlapping. Thus, it is possible that more than one sibling cases are eligible for adaptation at the same time. Selection policies [5] are therefore required to reconcile the situation and guarantee that there is only one adaptation case being activated at any specific point in time. We call such case the *active case* of the adaptation space.

Definition 4 (Adaptation Space). Let T denote a program component in a target web service. An adaptation space of T is defined as a labeled directed graph $G = (V, L, E)$ where V is a finite set of adaptation vertices of G; L is an ordered set of labels, each described by a set of use-conditions; E is a ternary relation on $V \times L \times V$, representing adaptation edges. For any α, $\beta \in V$ and $\ell \in L$, if $\alpha \Longrightarrow \beta$, then $(\alpha, \ell, \beta) \in E$. □

Given an adaptation space graph $G = (V, L, E)$, each vertex in V represents an adaptation case and each label in $(u, \ell, v) \in V \times L \times V$ represents a use condition describing when to switch off case u and turn on case v.

Example 6. Recall the four examples of adaptation cases for Q presented previously. The four cases are related with one another through the adaptation-of

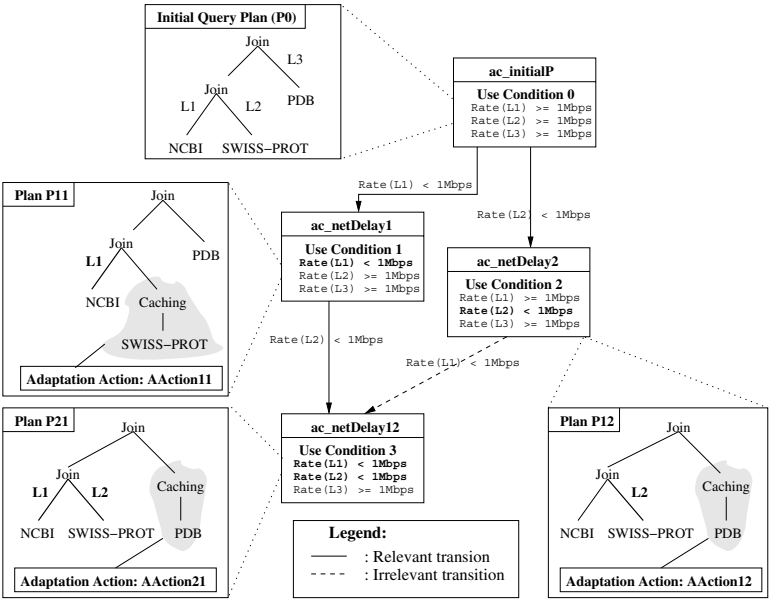


Fig. 8. Example of an Adaptation Space.

relationship, and together they form an adaptation space for Q, as shown in Fig. 8. Formally, this adaptation is described as a graph $G = (V, L, E)$, where

$V = \{ac_initialP, ac_netDelayL1, ac_netDelayL2, ac_netDelayL12\}$

$L = \{(L_1 < 1Mbps), (L_2 < 1Mbps)\}$

$E = \{(ac_initialP, L_1 < 1Mbps, ac_netDelayL1), (ac_netDelayL1, L_2 <$
$1Mbps, ac_netDelayL12),$
$(ac_initialP, L_2 < 1Mbps, ac_netDelayL2), (ac_netDelayL2, L_1 < 1Mbps, ac_netDelayL12)\}$

5 Issues of Using Adaptation Space Model

In this section, we summarize our discussion on the adaptation space model by classifying the concepts introduced into the following three categories: (1) what to adapt, (2) how to adapt, and (3) when to adapt.

What to Adapt. The first task we need to deal with in adaptation specification is how to identify the program components of a web service that should be adapted. We have addressed this issue by identifying what should be included in an adaptation context and what are the target units that we consider in our adaptation model.

When to Adapt. We have defined the use condition and the incremental approach for adaptation case specification in Sect. 3 and Sect. 4. A use condition is a set of predicates over some parts of the program state or adaptation space state, which hold over a period of time. It identifies when the corresponding adaptation case should be activated. Usually, a set of use conditions may not

become true all at once. Some use conditions may even be in conflict with each other. Therefore it is useful to be able to describe a partial order of adaptation contexts for a given program so that it can be adapted further as more use conditions become satisfied.

How to Adapt. The question of how a program component should be adapted involves the operational aspects of adaptation, which include issues such as how to monitor the use conditions, when to produce the adaptation code, how to apply (trigger) an adaptation, and how to integrate the adaptation code into the program.

In general, given an adaptation case, if its use condition can be evaluated at compilation stage, then we refer to the use condition as compile-time use condition and the adaptation case as compile-time adaptation. Otherwise, it is called runtime adaptation and its use condition is called runtime use condition.

In the runtime adaptation case, we need some runtime support, which detects when all the predicates in the use condition become valid, and when the adaptation is triggered to replace the target component. There are a number of ways to conduct situation monitoring. For example, we can use periodic monitoring with regular or irregular time interval [4].

Once the use condition becomes true, an immediate question to ask is when to apply adaptation. Both eager approach and lazy approach can be used here. When the use condition is evaluated to be true, the eager approach applies adaptation and replaces the component immediately, whereas the lazy approach only replaces the component at the time when the component is used.

For a runtime adaptation case, the runtime support must preserve the validity of the adaptation. This implies two basic tasks: (1) detecting when the predicates in the use condition are invalidated, and (2) replacing the adaptation component by an appropriate adaptation case whose use condition is satisfied.

6 Adaptation Space Model Application

We are currently developing, at Georgia Institute of Technology, a web service called Ginga [7,8], a self-adaptive query processing system, which has in its core the adaptation space model described in this paper. Ginga has two main phases: Proactive Engagement and Reactive Control. At Proactive Engagement, prior to runtime, Ginga generates an initial adaptation space for the input query. At runtime, Ginga Reactive Control phase monitors the use conditions. Whenever one or more use conditions become true, Ginga then navigates the adaptation space, selects the most appropriate adaptation case, and fires the associated adaptation action which replaces the current query plan with an alternative one optimized for the changes to the runtime environment configuration.

We have evaluated the benefits and trade-offs of adapting distributed queries using Ginga. Fig. 9 shows the comparative benefits of using Ginga versus no query adaptation when slow delivery is detected while downloading the remote data from the left-most operand (base relation called A) of a 7-way join query plan. All other base relations were also considered remote. The x-axis repre-

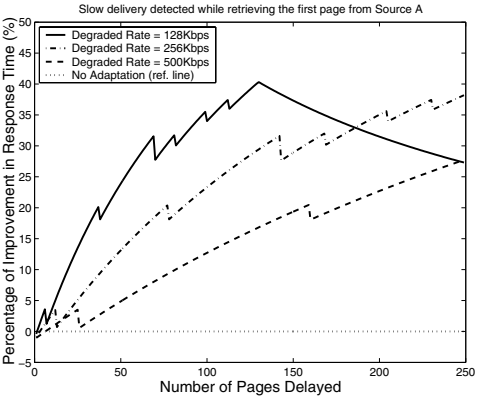


Fig. 9. Ginga adaptation engine performance: benefits of adaptation over no adaptation.

sents the number of pages from A that were delayed, and the y-axis represents percentage of improvement in response time obtained by using Ginga. Three degraded transfer rates were considered: $128Kbps$, $256Kbps$, and $500Kbps$. Normal transfer rate was $5Mbps$. As we can see from the graph, Ginga can provide up to 40% of improvement on the query response time when compared with the no adaptation case.

7 Related Work

The adaptation space based approach for program adaptation described in this paper can be seen as a seamless generalization and evolution of program specialization concept [1,12,13,11,6]. Program adaptation is a generalization of program specialization in both the objectives of adaptation and the methods of adaptation. Similar to program specialization, program adaptation presents an approach for adapting a generic program component to a given use context. However, *program specialization* is commonly viewed as a means for performance optimization. *Program adaptation* aims at a wider spectrum of enhancement of system capabilities, including performance, quality of service, and survivability. Program adaptation provides a uniformed framework for adapting programs to different levels of code specialization or capability relaxation (degradation) according to the various quality of service requirements and resource availability.

8 Conclusion

We have described the notation and the formal semantics of the adaptation space model used for designing adaptive web services. There are three main thrusts of the adaptation space approach. First, it defines a multi-dimensional adaptation context for capturing and coordinating different kinds of adaptation

at different levels of a web service. Second, it provides a uniformed way for representing and viewing a collection of alternative adaptations for a given query program or component of a web service. Third, it promotes a declarative and incremental approach to adaptation specification, allowing the incorporation of new adaptation behavior of a web service in terms of existing adaptation cases.

We have also described a web service called Ginga [7,8], which has in its core the adaptation space model described in this paper. Our experimental results show that Ginga query adaptation can achieve significant performance improvements (up to 40% of response time gain) for processing distributed queries over the Internet in the presence of end-to-end delays. We are currently extending the Ginga work to study other kinds of changes to system resource availability (e.g., CPU and memory constraints [9]) and their interactions with end-to-end delays and sudden unavailability of remote servers.

Acknowledgements. The first author was partially supported by CAPES - Brasilia, Brazil - and DoE SciDAC grant. The other two authors were partially supported by DARPA, DoE, and NSF grants.

References

1. C. Consel, L. Hornof, F. Noel, J. Noye, and E. Volanschi. A uniform approach for compile-time and run-time specialization. In *Partial Evaluation, International Seminar*, 1996.
2. C. Cowan, A. Black, C. Krasic, C. Pu, and J. Walpole. Specialization classes: an object framework for specialization. In *International workshop on object-oriented in Operating Systems*, 1996.
3. G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2), 1993.
4. L. Liu, C. Pu, W. Tang, J. Biggs, D. Buttler, W. Han, P. Benninghoff, and Fenghua. CQ: A Personalized Update Monitoring Toolkit. *ACM-SIGMOD'98*.
5. D. Maier. Adaptation spaces: Concepts and realization. Technical report, OGI CSE Technical Report, 1997.
6. D. McNamee, J. Walpole, C. Pu, C. Cowan, C. Krasic, A. Goel, P. Wagle, C. Consel, G. Muller, and R. Marlet. Specialization tools and techniques for systematic optimization of system software. *ACM TCS*, 19(2), 2001.
7. H. Paques, L. Liu, and C. Pu. Ginga: A self-adaptive query processing system. In *CIKM*, 2002.
8. H. Paques, L. Liu, and C. Pu. Distributed Query Adaptation and Its Trade-offs. In *ACM-SAC'03*.
9. H. Paques, C. Pu, and L. Liu. Query adaptation to changes in memory constraints. Technical report, Georgia Institute of Technology, 2003.
10. H. Paques, L. Liu, and C. Pu. The Adaptation Space Model. Technical report, Georgia Institute of Technology, 2003.
11. C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang. Optimistic incremental specialization: Streamlining a commercial operating system. In *SOSP*, 1995.
12. F. Tip and P. Sweeney. Class hierarchy specialization. *OOPSLA'97*.
13. E. N. Volanschi, C. Consel, and C. Cowan. Declarative specialization of object-oriented programs. *OOPSLA'96*.