A Framework for Relocation in Mobile Process-Centered Software Development Environments *

Supratik Bhattacharyya
Dept. Of Computer Science
University of Massachusetts
Amherst MA 01003, USA
(413) 545 3179
bhattach@cs.umass.edu

Leon Osterweil
Dept. of Computer Science
University of Massachusetts
Amherst MA 01003, USA
(413) 545 2186
ljo@cs.umass.edu

ABSTRACT

This paper addresses the problem of enabling a user of a process-centered SDE, hosted on a high-speed network, to continue working on a detached mobile workstation connected by a lower speed interruptable communications link. The focus of the paper is a process for determining whether and how to allow detachment. The process takes into account a broad range of factors, including the speed and reliability of the mobile link, the relative sizes and speeds of the mobile and networked workstations, the nature and state of the development process, and the importance of the detaching user. The paper presents a formal framework that allows us to specify the various factors, and an objective function that quantifies the inconvenience incurred by the detachment. The process uses the framework to determine which tools and resources to relocate when the user detaches from the network. A detailed example is used to illustrate this process.

Keywords

Process-centered SDEs, wireless, detachment, relocation.

Introduction

Wireless networking technology presents an opportunity to exploit mobile computing with portable devices to provide remote access to shared infrastructure. But this also gives rise to new technical challenges. For software development these challenges entail supporting collaboration and parallel development, as well as individual developer autonomy, for multiple participants who are widely dispersed geographically and are connected by a communication network. Past software development environments (SDEs) required continuous, reliable access to shared resources (eg. software process fragments, tools, files, etc.) via a high speed network. With wireless networking users can operate in one of two modes - **attached mode**, where connectivity is provided by high-speed wired links (eg. a LAN) , or **detached mode**, with connectivity through a wireless link. The bandwidth of a wireless link is about 1-2 Mbits/s, whereas high-speed fiber-optic links can provide a 155 Mbits/s bandwidth. In addition, a wireless link is error-prone and can suffer from frequent disconnections. Also, mobile computers can be expected to have lower processing power and storage capacity than workstations.

Approaches to compensating for poor network access quality include pre-fectching of certain files and software tools needed during a planned period of detachment, and work redistribution to avoid having the mobile user become a bottleneck. These approaches seem applicable to all domains where wireless technology is used. In this paper we explore the application of such approaches for the specific domain of Process-Centered SDEs (PSDE's), and show how the explicit process representations found in PSDE's enable us sharpen these approaches.

*This work was supported in part by the Air Force Materiel Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract F30602-94-C-0137 and in part by the National Science Foundation under grant NCR-95-08274.

Approach to this work

In this work we present a framework for evaluating and handling PSDE user requests to detach from a wired network and go mobile. We characterize some consequences of this, identify factors needed to quantify the detachment scenario, suggest a cost function whose optimization characterizes the ideal reaction to the request, and suggest how to optimize the cost function.

Our focal point is the objective function, which quantifies user inconvenience, and a method for minimizing it. We suggest how to decompose the objective function into subfunctions quantifying different types of inconvenience. We show how the factors characterizing the scenario are used to define these subfunctions. We also suggest how and why the PSDE process might be mod-

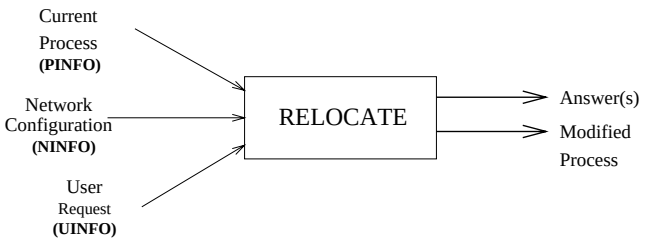

Figure 1: A high-level view of decision framework

ified, and how tasks might be reassigned to users, in order to minimize the objective function that measures inconvenience.

Figure 1 is a schematic of our work, centering on the role of the RELOCATE relocation request analysis engine. RELOCATE inputs three types of data describing the detachment scenario, uses it to compute the objective function for all ways of honoring the detachment request, and outputs a recommendation for how to honor the request. The output may be either a denial of the request or a suggestion of how to modify the process, prefetch tools and data, and assign upcoming steps to the detaching user. We now present details of the features shown in Figure 1.

Inputs to RELOCATE

The input to RELOCATE is three types of information, PINFO (Process Information), NINFO (Network Information), and UINFO (User Request). Each is now described in detail.

PINFO : process information needed
An SDE is a collection of tools, a repository for software resources like tools, files, code, etc. and a set of data and management policies. A PSDE also provides a representation of the software process itself. The representation may be by a model such as a Petri Net or flowgraph, or by executable code written in a coding language. [8].

PINFO characterizes the PSDE process representation with both static and dynamic information. Static information expresses the process structure. Dynamic information represents process execution state.

Static Information :
We assume that a PSDE process consists of a set of *steps* to be executed procedurally or as triggered reactions to events. A step, may contain substeps or **tasks** to be performed either in sequence or parallel. A task can contain a set of preconditions to be satisfied, a structure of substeps, a set of resources potentially required, a set of postconditions potentially fulfilled, a set of reactions potentially triggered, and a set of artifacts to be produced. While one cannot know beforehand which resources will be actually used, which reactions will be actually triggered, and which postconditions will be actually satisifed, it suffices to determine supersets statically by making simplifying assumptions.

Dynamic Information :
Dynamic process information describes the current state of execution of the process and changes with time. Examples of the pieces of information that determine the dynamic state of the process are: the step(s) currently being executed, the status of locks on shared resources, software tools currently in use, ongoing collaborative activities, and values of variables/objects stored in repositories.

NINFO: network information needed
We assume SDE communication is provided by a heterogenous wired/wireless network. The backbone is a high-speed network of arbitrary topology. Users connect to the backbone by high-speed links in attached mode and low-speed interruptable wireless links in detached mode.

To simplify matters initially, we assume a star topology for the network, where the star's central node represents the entire wired part. All stationary users attach to this center. Software resources can be *physically* located anywhere on the backbone, but because all are interconnected by high-speed links, we abstract their location to a single *logical* site. A mobile (detached) user is sited at the end of an arm of the star, connected by an edge representing a link that may be wired or wireless. This encourages a focus on communication over the star's arms.

The specific information needed to describe the network are :

- **The bandwidth provided by the wireless channel (B_w)** and how that compares to the bandwidth (B_h) of the high-speed wired links. We denote the ratio of B_w to B_h by B_{fr}. In a cellular network, all cell users share available bandwidth. Thus available bandwidth varies over time.
- **The reliability of the wireless link.** Wireless links will go down suddenly and stay down unpredictably, requiring users and processes to adapt. Link reliability is characterized by two factors : the expected fraction of time during which the link will be down (L_{exp}) and the frequency of disconnection (N_{exp}). Both will change with time.
- **The storage capacity/processing power of the mobile computer.** Mobile computers are de-

signed for portability and low power consumption, and can be expected to have reduced storage capacity/processing power. RELOCATE should know the mobile computer's storage capacity (St_{Max}) and processing power CPU_{mob} so that it does not try to overload the mobile computer. Available storage space will also change over time.

UINFO : user request
The user's detachment request (UINFO) consists of:

- **Estimated time before detachment.** This is needed to help RELOCATE decide what it can possibly relocate. Thus, if time to detachment is short, very little can be downloaded.
- **The expected period of detachment (T_{detach}).** RELOCATE needs this to predict the process steps that may be executed while the user is detached.
- **The user profile.** This helps the process decide the urgency of the request. Requests from "important" users may be allowed to inconvenience others significantly.
- **Suggested activities during detachment.** This helps reassure RELOCATE that projections of user activities during the detachment period are realistic.

The Objective Function : Identifying User Inconvenience

PSDE developers/users should be unaffected by detached operation to the greatest extent possible. Thus it seems reasonable to define the discernible adverse impact of detachment to be a cost function to be minimized by the RELOCATE engine. We refer to this adverse impact as **user inconvenience**. The first step in defining user inconvenience is to identify a number of cost factors, each corresponding to a different type of inconvenience. Our goal is to formulate for each factor i the inconvenience cost $COST_i$.Then the total inconvenience cost INC_COST is given by some weighted sum of all the costs :

$$INC\_COST = \sum_i w_i * COST_i$$

The cost attributable to each factor can, in turn be viewed as a function of the cost to two different entities :

1. The mobile user U who **may** be in the detached mode.
2. The rest of the development team, collectively referred to as G.

If for $COST_i$, we denote inconvenience for U as UC_i and inconvenience for G as GC_i, then $COST_i$ can be expressed as :

$$COST_i = F(UC_i, GC_i)$$

Cost Factors

We can now identify some of the factors that seem most likely to be the leading sources of inconvenience and define the cost function for each.

1. <u>Resource transfer</u> : This cost arises from the need to transfer resources to and from the detached user's computer. We must evaluate the costs due to downloading resources to the mobile user prior to detachment, uploading artifacts produced at the mobile user, delays in receiving notification about the release of locks on files, etc. The transfer time for resource R can be expressed as a function of its size , the available bandwidth and the link characteristics as :

$$TX\_TIME(R) = fn1(size(R), B_{fr}, N_{exp}, L_{exp})$$

where $size(R)$ denote the size of a software resource required to be downloaded for activity A. Now let $Delay\_Cost(A, t)$ measures the cost incurred due to a delay of time t in starting the execution of A. Then suppose that before activity A starts, all the resources that it needs have to be transferred. Suppose N resources $R_1, ..., R_N$ need to be dowloaded over the wireless link after the user detaches, Then the inconvenience cost is

$$S_1 = \sum_{i=1}^{N} Delay\_Cost(A, TX\_TIME(R_i)).$$

Let $P_1, ..., P_m$ be the activities that are potentially affected by the delay in the transfer of artifacts produced by activity A and by delays in receiving needed notifications from activity A. These delays can also be expressed in terms of the TX_TIME and a function NOTIFY_DELAY, defined as

$$NOTIFY\_DELAY(P_i) = fn2(P_i, B_{fr}, N_{exp}, L_{exp})$$

for the delay suffered by activity P_i due to the delay in receiving notification(s) from A. The total inconvenience cost all such delays is :

$$S_2 = \sum_{i=1}^{m}[Delay\_Cost(P_i, NOTIFY\_DELAY(P_i)) + \sum_{k=1}^{K} TX\_TIME(R_{i,k})]$$

where $R_{i,1}, R_{i,2}...R_{i,K}$ are the resources needed from the detached user in order to perform P_i.
S_1 and S_2 are measures of inconvenience to the detaching and remaining user (U and G respectively). UC_{ResTx} and GC_{ResTx}, the costs for U and G, will be determined by where activity A is assigned. For example, if A is assigned to U then $P_1, ..., P_m$ are activities at G that are delayed by A. In that case,

$$UC_{ResTx} = S_1 \text{ and } GC_{ResTx} = S_2$$

The total resource transfer cost $COST_{ResTx}$ is some function(eg. the simple sum) of UC_{ResTx} and GC_{ResTx} :

$$COST_{ResTx} = F_{ResTx}(UC_{ResTx}, GC_{ResTx})$$

2. **Storage constraint** : This is the cost arising from the inconvenience to users caused by limited storage capacity on a mobile computer. Excessive storage requirement for the resources necessary for an activity may prevent the execution of the activity at the mobile user site. This may result in the activity being reassigned to a stationary user or not being executed at all during the period of detachment.
Let an activity A require resources $R_1, R_2, ...R_N$, and let $P_1, P_2, ...P_M$ be the artifacts produced. Then the total storage requirement for A is

$$V = \sum_{i=1}^{N}(Size(R_i)) + \sum_{i=1}^{M}(Size(P_i))$$

Then we can define a function $ST\_Cost(A)$ for the storage cost due to A :

$$ST\_Cost(A) = h(V)$$

The form of h() above can vary from one SDE to another. One possible form is to make the cost linear in x below a certain threshold and very high above that, eg.

$h(x) = Cx, x < Thresh,$
$\quad = \infty,$ otherwise.

With a more sophisticated function, the constant C can be replaced by a function of the available storage space on the mobile computer at the time of execution of A.
Evidently, $ST\_Cost(A)$ is the inconvenience cost to the detaching user U and does not affect G. Hence in this case,

$$UC_{St} = ST\_Cost(A) \text{ and } GC_{St} = 0$$

and the total cost is

$$Cost_{St} = F_{St}(UC_{St}, GC_{St})$$

3. **Processing Bottleneck** : Mobile computers can be expected to have lower processing power. Executing computation intensive tasks may not only affect the mobile host but also other users by delaying the satisfaction of preconditions (eg. release of locks on software resources) that these users may be waiting on.
Let $T_{diff}(A)$ be the additional time taken to execute an activity A on a mobile computer. Clearly $T_{diff}(A)$ is some function of the relative execution and input/output speeds of the detached and the attached computers.Let $P_1, P_2, ..., P_m$ be the tasks assigned to U that have to await the completion of A. Let $Q_1, Q_2, ..., Q_n$ be the tasks assigned to G that have to await the completion of A. In order to account for the inconvenience to the user (as opposed to the delay in executing other tasks) due to the slow execution of A, we define $SLOW\_COST(T_{diff}(A))$. The cost UC_{CPU} for the detaching users is given by

$$UC_{CPU}(A) = SLOW\_COST(T_{diff}(A)) + D$$

where

$$D = \sum_{i=1}^{m} Delay\_Cost(P_i, T_{diff}(A))$$

The cost GC_{CPU} for the remaining users is given by

$$GC_{CPU}(A) = \sum_{i=1}^{n} Delay\_Cost(Q_i, T_{diff}(A))$$

As before, the total inconvenience cost for activity A is *some* function of these two costs :

$$COST_{CPU} = F_{CPU}(UC_{CPU}, GC_{CPU})$$

4. **Collaboration overhead** : Any software process step involving collaboration among developers (eg. a review session) requires use of the limited bandwidth interruptible wireless channel. Temporary disconnections of the channel may result in delay in information reaching U from G and vice-versa, thereby causing inconvenience to all the participants.
Leaving the detaching user out of the collaborative activity would eliminate the delays but may result in other types of inconveniences to both U and G. Hence the form of the cost function will depend on the decision to incude or exclude U.
Let$B_{Req}(A)$ be the required bandwidth for a collaborative activity A and $T_{duration}$ be the expected duration. When U is included, we can define a function $GC_{COL}(A)$ for the inconvenience to G as :

$$GC_{COL}(A) = fn3(T_{duration}, B_{req}, B_w, N_{exp}, L_{exp})$$

and the cost to the detaching user as

$$UC_{COL}(A) = fn4(T_{duration}, B_{req}, B_w, N_{exp}, L_{exp})$$

Let us define functions $U\_EXCLUDE(A, UP)$ and $G\_EXCLUDE(A, UP)$ to measure the inconvenience cost to U and G respectively when U is excluded from A. The term UP represents the user profile/importance. Then we have

$$GC_{COL}(A) = G\_EXCLUDE(A, UP)$$

and

$$UC_{COL}(A) = U\_EXCLUDE(A, UP)$$

In either case, the the total inconvenience cost is

$$COST_{COL} = F_{COL}(UC_{COL}, GC_{COL})$$

A collaborative activity may require running one or more software tools. For each tool there are processing and storage considerations. The inconvenience costs for these are accounted for separately as processing bottleneck and storage constraint costs.

5. **Relocation Overhead**: Costs are also incurred due to the reassignment of a task in order to alleviate the problems mentioned above. When a task is reassigned to a user, then it may delay some or all of the other tasks that have already been assigned to that user. In this sense, the user is inconvenienced by the user who was originally supposed to execute the task.
   Let activity A of duration τ_A be *reassigned* to a user. Then suppose there are n other activities $P_1, ..., P_n$ at the user that get delayed by this reassignment. Then we can express the total cost of this reassignment as the following function :

   $$RELOC\_COST(A) = \sum_{i=1}^{n} Delay\_Cost(P_i, \tau_A)$$

   If the reassignment is to U, then

   $$UC_{Reloc} = RELOC\_COST(A) \; and \; GC_{Reloc} = 0,$$

   else

   $$GC_{Reloc} = RELOC\_COST(A) \; and \; UC_{Reloc} = 0.$$

Other cost factors can be defined as well. For any detachment considerations, it may be desirable to take into account some or all of the above as well as some additional costs. In the next section, we describe how RELOCATE uses whichever of the factors that the end-user selects.

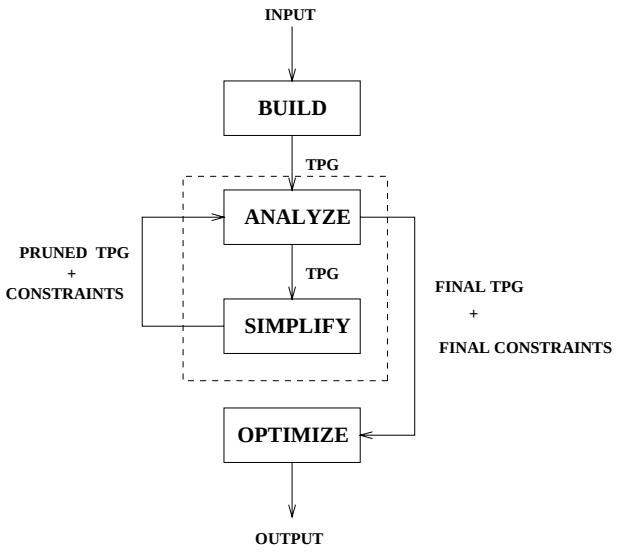

Figure 2: RELOCATE engine stages

Structure of the RELOCATE Engine

RELOCATE is the engine for the decision making framework. Given the necessary input information, it reaches its decisions via the flow graph in Fig 2. The flow graph is represented as three modules, of which the second one is further divided into two sub-modules. A high-level description of their functionality is provided below :

BUILD :

For the given process or process fragment (represented by PINFO), this module builds a *task precedence graph*(TPG) to represent the precedence relation among tasks in the process fragment. Assuming that all cycles in the TPG are unrolled, the TPG can be reduced to a tree. The details of the TPG and its constructions are described in a later section.

ANALYZE-SIMPLIFY :

The ANALYZE module scans the TPG and evaluates the difficulty or complexity of actually answering the questions posed to RELOCATE. If the decisions are deemed "too difficult", then the routine SIMPLIFY is invoked to perform simplifications.
The SIMPLIFY module will apply some predefined policies/heuristics specific to the SDE under consideration. One type of simplification is the pruning of the task precedence graph. Another simplification is the (possible) generation of conditions/constraints that will reduce the number of alternative decisions that RELO-

CATE has to choose among in the OPTIMIZE routine. The ANALYZE-SIMPLIFY cycle may be iterated a number of times till ANALYZE decides that the graph has been reasonably simplified. The exact criteria for evaluating the complexity depends on the size of the input graph, the input information provided and the design of RELOCATE itself. It will have to be readjusted through experience with RELOCATE.

OPTIMIZE :
This module weighs the various alternatives in response to the UINFO request and selects the "best" solution. It performs this by optimizing the objective function, subject to certain constraints. The constraints may arise from various sources - the structure of the task precedence graph, conditions generated by the SIMPLIFY routine and user requests.

The RELOCATE engine is designed to be rich enough and flexible enough to solve a range of problems with varying degrees of complexity. For example, it can provide "yes/no" answers to relatively simple questions such as whether a user will be allowed to download a single data file. On the other hand, it can be given an entire software process and asked to come up with an "efficient" allocation of process fragments to users - in effect, producing a new, modified version of the software process. It can be asked to solve problems dealing with specific aspects of mobile computing, eg. what if a user is going to use a laptop computer with low processing speed.
It is to be noted that the RELOCATE engine goes beyond performing resource relocation based on the process fragment that a user is likely to execute when detached. When RELOCATE receives a request from a detaching user it makes a decision as to whether the request should be honored and to do so, it may tailor the entire software process before relocating resources. Thus, its decisions are aimed at providing "global benefits" within the SDE as opposed to providing benefits to a single user. This is provided for by the design of the cost function.

Example

At this point, we introduce a small fragment of a software design process that will be used to used to illustrate and explain each of the modules of RELOCATE in Fig 2 and various of the cost function computations of Section 3. The example has been drawn from the well-known Booch Object-Oriented Design(BOOD) Process. The process fragment is a small, simplified segment of step 1 of BOOD - a team of designers collaborating to identify a set of objects from the requirements specifications provided. The requirement specifications are assumed to be clearly divided into a number of subsections. Each designer is assigned one or more subsections and each comes up with a list of objects for the assigned section(s). However, the object names must be unique across all the lists. Hence the designers need to have some form of interaction to resolve the name conflicts. In reality, this name conflict check may be applied every time a designer comes up with a new object. But we make the reasonable assumption that each designer comes up with a complete list, and after that, name conflict resolution is applied across all these lists.

We will identify the task/resource allocation decisions that the process wishes to make when a member of the design team desires to switch to a detached mode of operation. Then we will demonstrate what RELOCATE does to reach its recommendation of which tasks to assign to which user.

The BUILD module

Building a Task Precedence Graph
The task precedence graph(TPG) is a representation of a subset of the software process tasks and their partial ordering, rooted at the currently executing task(s). Note that this information is provided as part of PINFO. The nodes in the graph represent tasks still to be done and the directed edges represent their ordering. The TPG can be thought of as an unrolling of the graph representation of the original process by unrolling all loops a (finite) number of times sufficient to anticipate all iterations that are expected during the proposed detachment period. In addition, if there are multiple users collaborating on the execution of a task, then the TPG represents the collaboration by creating multiple copies of the task, one for each user. Thus the TPG contains special fork and join nodes to represent this parallelism. An edge emanating out of a (non-fork) task represents a transition to another task that may immediately follow it. Each such edge can be labelled with the probability that it will be taken after the execution of the task. Thus the sum of the probabilities over all the edges out of a given (non-fork) node is 1. The sum of the probabilities of all the edges leaving a fork is greater than 1. We assume that each task T in the precedence graph has a probability value P(T) associated with it. This is the probability that the task will begin execution during the period of detachment. Once a task begins execution, then the probability that it will finish execution before the end of the detachment period is

$$P_f(T) = Prob(\tau_T <= T_{detach} - \sigma_T)$$

where τ_T is the expected execution time for the task and σ_T is the expected starting time, relative to the beginning of the period of detachment. This is a function of the expected execution times of all ancestors of T.
If a task T has only one predecessor T_{pred}, then clearly

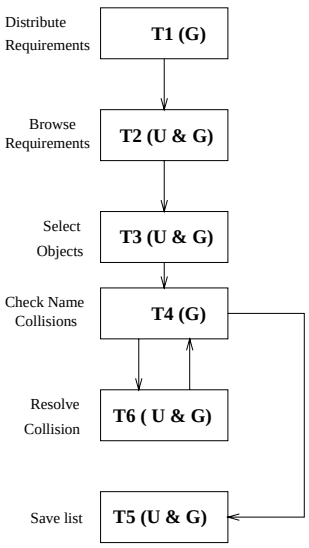

Figure 3: Flow graph for two-user (U & G) collaboration on a design process fragment

$$P(T) = P(T_{pred}) * P_f(T_{pred}) * P_E((T_{pred}, T))$$

where (T_{pred}, T) is the edge representing the transition from T_{pred} to T and $P_E((T_{pred}, T))$ is the probability that T is executed after T_{pred} terminates. If a task has n predecessors (eg. the termination of a CASE statement), T_{predi}, i = 1, 2 ...n, then we must estimate P(T) as

$$P(T) = max_{i=1}^{n}[P(T_{predi}) * P_f(T_{predi}) * P_E((T_{predi}, T))]$$

If T is a join node, then with n predecessors 1, 2, ... n, then

$$P(T) = \prod_{i=1}^{n}[P(T_{predi}) * P_f(T_{predi}) * P_E((T_{predi}, T))]$$

Now, if we assign a probability of 1 to the task at the root of the TPG, and probabilities to all edges of the TPG, we can compute P(T) for every task T, if we are also given all τ_T for all T and T_{detach}.
All tasks in the graph may not be equally valuable from the point of view of the software process. Hence a value $V(T)$ needs to be associated with every task T. Again, from the point of view of the relocation process, all tasks may not be given the same weight and then a weight $W(T)$ can be associated with each task T, given by :

$$W(T) = V(T) * P(T)$$

Task Precedence Graph for BOOD Process Fragment
Fig 3a represents the flow graph for the design process fragment described earlier. Note that the graph contains a loop. Therefore in Fig. 3b, we show the corresponding TPG for the fragment, rolled out to represent one iteration.
Note also that Fig 3a. represents a collaborative design process that should be carried out by two users U & G. Some of the tasks are to be executed by both U and G. Hence in Fig 3b., two copies of each of such tasks are used to represent the collaboration by U and G. For example, task T3 in Fig. 3a is represented by two tasks - T4 for U and T5 for G in the TPG of Fig. 3b. to represent collaboration by U and G in the "Select Object" activity.
Assume now that the requirements specifications are divided into two parts : **Req_Spec1** assigned to U and **Req_Spec2** assigned to G. Tasks T2 and T3 represent the activities of U and G browsing through their respective parts. Objects selected by U are added to a list **L1** (task T4). Objects selected by G are added to a list **L2** (task T5). However, the name collision check across the two lists is to be performed by only one of the two users (in this case, G). This is represented by tasks T6. Once the check is completed, one of two cases can arise :

- no collisions are found and the lists can be written out to a repository (tasks T7 and T8).
- collisions are found. In that case, U and G have to make modifications to their lists (tasks T9 and T10) to resolve the collision(s). After that, G has to perform the check once more (task T11). Only two levels of this check-resolve iteration are represented in Fig. 3b.

Our example application of RELOCATE will now address the question of how to deal with a request that U be allowed to detach prior to the start of this process fragment, i.e. prior to the start of T1. Some possible alternatives for RELOCATE are :

OPTION 1 :
To let U take away the process fragments represented by the tasks marked with U in Fig. 3b. In that case, the following resources must be downloaded to U prior to detachment :

1. The requirement specifications Req_Spec1.
2. The (possibly empty) list L1.
3. A browsing tool.
4. An object entry tool for adding objects to L1.

The inconvenience cost for choosing this option includes the following factors :

1. Storage constraint ($Cost_{St}$).

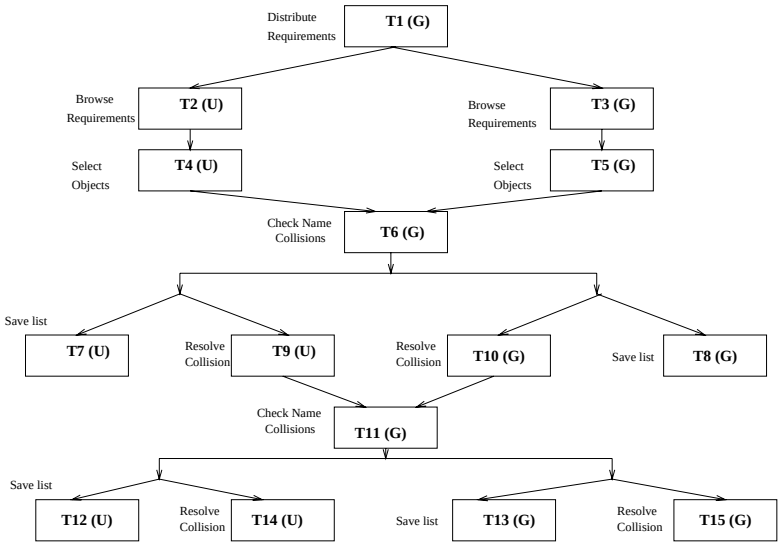

Figure 4: Task precedence graph for design process fragment

2. The difficulty faced in downloading L1 for the name collision check at G and then communicating the feedback information to U (Resource Transfer cost $COST_{ResTx}$).

3. The overhead of collaboration by U and G in resolving name collisions (Collaboration Overhead cost $COST_{Col}$).

4. The difficulties in running the browser and the object entry tool on the (possibly slow) mobile computer (CPU bottleneck cost $COST_{CPU}$).

OPTION 2 :
To reassign all of U's tasks to G. No resources have to prefetched to U in this case. However, this puts additional workload on G (Relocation Overhead cost $Cost_{Reloc}$).

OPTION 3 :
To refuse to let the detaching user U execute any task till it re-attaches. This will cause inconvenience at U due to the postponment of all the tasks that are assigned to U and hence the creation of list L1. This will also restrict G from performing certain tasks, because collision can be checked and resolved only after both the lists L1 and L2 have been created. If T_{detach} is the period of detachment then for each task A that is postponed, the assigned user will incur a cost $Delay\_Cost(A, T_{detach})$. The total inconvenience cost is the sum of the costs for all the postponed tasks.

RELOCATE works by computing the cost for all such options and then choosing the one that incurs the minimum cost.

Task Id(T)	Input Edge Probabilities	P(T)
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	1.0	1.0
5	1.0	1.0
6	1.0, 1.0	1.0
7	0.8	0.8
8	0.8	0.8
9	0.2	0.2
10	0.2	0.2
11	1.0, 1.0	0.2
12	0.8	0.16
13	0.8	0.16
14	0.2	0.04
15	0.2	0.04

Table 1: Edge Weights and Task Probabilities

In order to compare the options, W_is have to be computed for each of the tasks, as described earlier. In this example, we assume that $W(T)$ values for the tasks are computed under the following assumptions :

1. $V(T) = 1.0$, for all T.

2. $\tau(T) = 1.0$ for all T.

3. $T_{detach} >> \tau(T)$ for all T.

4. The probabilities on the various edges are as shown

in Table 1.

The OPTIMIZE module

Objective Function definition

In order to optimize the objective function INC_COST, we assume that for each of N nodes in the TPG, denoted by T_i, i=1,...,N, we have the following associations :

1. A probability P_i and weight W_i as described before.
2. A variable x_i such that :

   $x_i = 0$, if the task is **not** executed during detachment, resulting in the non-execution of all tasks in the DAG rooted at this task.
   $= 1$, if the task is executed at the central site.
   $= 2$, if the task is executed at the remote site.

3. Costs $UC_i(x_i), (x_i = 0, 1, 2)$, representing the costs incurred for U for the values of x_i.
4. Cost function $GC_i(x_i)$, $(x_i = 0, 1, 2)$, representing the costs incurred for G for the values of x_i.

Then for each vector $X_\alpha = (x_{\alpha 1}, x_{\alpha 2}, ..., x_{\alpha N})$, where $x_{\alpha i} \in \{0, 1, 2\}, \forall i = 1, ...N$, we can define the $UC(X_\alpha)$, $GC(X_\alpha)$ and $INC\_COST(X_\alpha)$ as :

$$UC(X_\alpha) = \sum_{i=1}^{N} W_i * UC_i(x_{\alpha i})$$

$$GC(X_\alpha) = \sum_{i=1}^{N} W_i * GC_i(x_{\alpha i})$$

$$INC\_COST(X_\alpha) = W1 * UC(X_\alpha) + W2 * GC(X_\alpha)$$

where W1 and W2 are constants that weigh the relative importance of U and G, specified as part of UINFO. Our aim is to determine the vector X^* such that

$$INC\_COST(X^*) = MIN(INC\_COST(X_\alpha).$$

over all $X_\alpha \in \{0, 1, 2\}^N$). This minimization must however be subject to constraints on the values of the x_is. These constraints come from a variety of sources :

- **Precedence relations among tasks in the graph.** For example, if task T_i precedes task T_j, then all vectors for which $x_i = 0$ *and* $x_j = 1$ or for which $x_i = 0$ *and* $x_j = 2$ (i.e. task T_j executes though its predecessor is abandoned), are not feasible. In order to ensure that no such vector is found to minimize INC_COST, the optimization process must be constrained. The constraint $2x_i - x_j \geq 0$ suffices to assure the desired results.
- **PSDE-imposed requirements** As another example, suppose that RELOCATE UINFO specifications mandate that tasks T_i and T_j must always be executed at the same location (eg. the same developer). Then any feasible solution will also have to satisfy the constraint $x_i - x_j = 0$.
- **SIMPLIFY module constraints**. In the next section we will see that such factors as storage requirements may prevent the assignment of a task to a detaching user. This can be easily modelled by the constraint $x_i \leq 1$. This and other similar restrictions are generated by the SIMPLIFY module as part of the simplification of the optimization computation overhead.

With N tasks and 3 possible values for each x_i, there can be as many as 3^N different assignments of values for vector X. Minimizing the INC_COST function, subject to the constraints, will yield a feasible optimal vector X (and consequently an optimal assignment of tasks to users). This will determine which process fragments should be executed at the remote site during detachment. Accordingly, resources required for these should be downloaded to the user before detachment.
Of course, whether all of them can be downloaded depends on the time remaining before detachment (information provided by user as part of UINFO). If it is not possible to download all resources, some arbitration policy would have to be used. For example, priority might be given to downloading resources that will ensure the execution of certain key tasks. The process for executing this downloading is beyond the scope of this paper. Computation of $UC_i(x_i)$s and $GC_i(x_i)$s is based on the cost functions described earlier and depends on which of the cost factors we choose to consider. For example, we may consider only the resource transfer cost and the collaboration overhead cost for a given query/request. In that case, the costs due to all other factors (eg. CPU overhead cost) are set to zero. However we make a set of general assumptions :

1. There is an initial task assignment (i.e. a set of assignments to x_is) $X_{attached}$ that represents how the tasks are assigned to U and G when U is operating in the attached mode. Let $x_{i\,att}$ be the value assigned to x_i under $X_{attached}$.
2. For any assignment $X = \{x_1, ..., x_N\}$, if x_i is 0 (i.e. the task T_i is not executed at all during the period of detachment, the cost incurred by the developer (U or G) who was initially assigned the task (under $X_{attached}$) can be expressed as $Delay\_Cost(T_i, T_{detach})$, where T_{detach} is the period of detachment. This cost is added to $UC_i(x_i)$ or $GC_i(x_i)$ accordingly.

The ANALYZE-SIMPLIFY module

In the SIMPLIFY module, predefined guidelines, based both on past experience and special requirements of the PSDE under consideration, are used as heuristics to create constraints used in the OPTIMIZE step to reduce the computational effort. Some examples are :

1. *If the probability value P_i associated with a task is less than a certain threshold* ***Thr*** *then prune the subtree rooted at that task node.* The effect is to prevent the execution of these tasks while the user is detached. This effects a process modification.
2. *Exclude the mobile user from some or all collaborative activities during the detachment period.* This will depend on a number of factors - the importance of the user in the collaborative activity, the disconnectivity characteristics i.e. $N_{exp} and L_{exp}$, the bandwidth available i.e. B_w and the bandwidth required for the activity.
3. *If the user has expressed a preference about activities during detachment, then honor the preferences.* The extent to which the relocation process honors those preferences can depend on a number of factors eg. the importance of the user.

The benefits of following such guidelines are two-fold :

- Pruning the task precedence graph based on **Thr** (example 1 above) can reduce the number of tasks to be considered in the optimization step and can therefore simplify the optimization computation.
- A number of restrictions (eg. examples 2 and 3) above can be easily modelled as constraints on the cost function to be optimized and can reduce the number of feasible solutions to be considered by the optimization step.

The simplification guidelines are organized in levels. When the SIMPLIFY module is invoked the first time from the ANALYZE module, it applies the simplification guidelines at level 1. The next time it is invoked it applies guidelines at level 2 and so on. For example, at level 1, all tasks with probability of execution less than 0.01 may be pruned while at level 2, the cutoff probability may be set to 0.05.
ANALYZE decides the extent to which simplification is necessary. It can base its decision on a number of factors. For example, the computational difficulty of the optimization increases with the number of tasks in the TPG. ANALYZE may repeatedly invoke SIMPLIFY (as in the above example) till the number of tasks is sufficiently reduced. Again, consider a query like the following : Does the mobile computer have sufficient capacity to store a certain file? In this case the network connection, CPU speed, etc. are not relevant factors. ANALYE will then invoke SIMPLIFY to set the costs $COST_{ResTx}, COST_{CPU}, COST_{COL}$, etc. to zero ensure that the decision is not influenced by any of these factors.
As an example of heuristic pruning, let us consider Fig. 3b. If we apply heuristic 1 with Thr set to 0.2, then it enables us to prune the tasks T12 through T15 to get the graph in Fig. 4. This is the graph we consider now to complete our example.

Optimization Example

The cost computation and optimization technique will now be applied to the pruned graph in Fig. 4. It has 11 tasks T1 to T11 and the precedence relation among them gives rise to a large number of constraints on the values of x_i. Since these are straightforward, we do not list them here. However, let us assume the existence of some PSDE-imposed restrictions and design the constraints to model them :

1. task T1, T3 T5, T6, T8, T10, T11 have to executed at the central site. This implies that :

$$x_j = 1, \text{j} = 1,3,5,6,8,10,11$$

2. All of the tasks T2, T4, T7, T9 have to be executed at the same site or all of them have to be postponed till U reattaches. ie.

$$x_2 = x_4 = x_7 = x_9$$

With the above restrictions, we have heavily restricted the relocation options. The decision that remains to be made is whether to allow tasks T2, T4, T7 and T9 to be executed by the detaching user U, to reassign them to G or to postpone their execution altogether. Of course, a postponment will imply, that tasks T6, T8, T10 and T11 (assigned to G) also have to be postponed (due to the precedence constraints). We already have $x_j = 1$, j = 1,3,5. For the remaining variables, there are only three feasible assignments of value sets :

1. $x_2 = x_4 = x_7 = x_9 = 0$; $x_6 = x_8 = x_{10} = x_{11} = 0$.
2. $x_2 = x_4 = x_7 = x_9 = 1$; $x_6 = x_8 = x_{10} = x_{11} = 1$.
3. $x_2 = x_4 = x_7 = x_9 = 2$; $x_6 = x_8 = x_{10} = x_{11} = 1$.

The inconvenience cost factors for these three choices have already been described in Section 4.1.2. It is easy to see how the various cost factors influence the optimal choice. For example, if $COST_{Reloc}$ is very low (i.e. U's task can be reassigned to G without causing too much inconvenience to G), then option 2 will be the optimal choice. On the other hand, if the other costs like

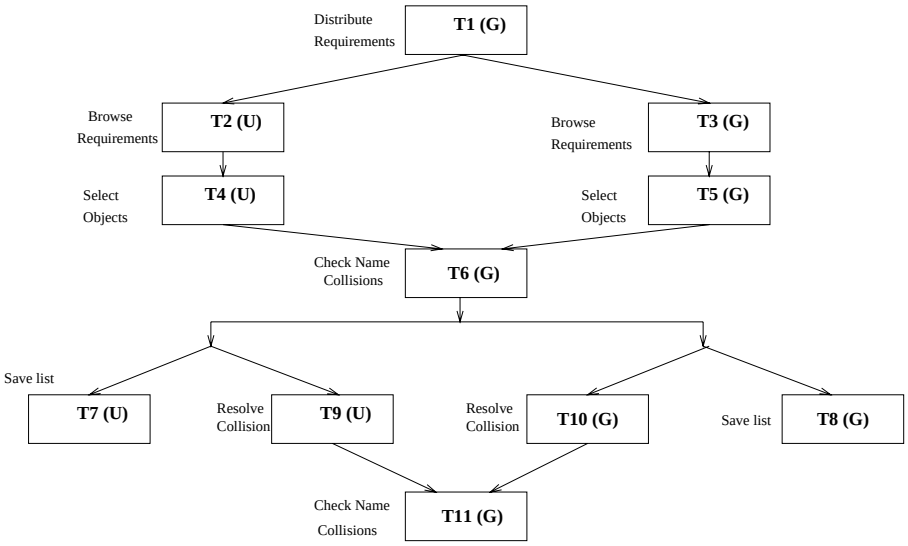

Figure 5: Pruned task precedence graph

$Cost_{ResTx}$ etc. corresponding to option 3 (ref. Section 4.1.2) are low, then option 3 is optimal and U is allowed to take away the process fragments represented by T2, T4, T7 and T9.

Related Work

The issues raised by the mobile computing paradigm include dealing with the problems of low bandwidth, disconnection, conservation of battery power for mobile hardware and storage limitations. These issues have been discussed in [1]. The adverse effects of these can be reduced by minimizing the communication requirements between the mobile hardware and wired backbone network. One approach is to have "user agents" inside the wired part of the network, acting on behalf of a mobile user. This has been proposed in several contexts([2, 3]). If we consider each user/developer in a multi-user SDE as an already existing"agent", then the reassignment of tasks from the detaching users to stationary users is essentially the application of the "user agent" paradigm in the SDE context.

Disconnection from the network can be either planned or unplanned. Unplanned disconnection refers to the intermittent loss of connectivity while the user is still attached to the network over a wireless link. Planned disconnection is the case where a mobile user disconnects completely from the network to operate autonomously. The Coda file system [4] has been designed to make network disconnections transparent to the user, by creating on-board file caches and by devising techniques for maintaining consistency of shared data. [5] discusses the issue of consistency in SDEs where there is more flexibility than in traditional databases systems. The paper also discusses the issue of "lazy writeback", which can be beneficial for a more efficient use of bandwidth when it is an expensive resource, eg. in the case of a wireless link. Our work does not aim to address the issue of maintenance of consistency but simply assumes that there is *some* underlying mechanism for doing it. However, we do address the issue of prefetching data prior to detachment to minimize the adverse impact of the wireless channel on users.

The issue of disconnected and low-bandwidth operation in the specific context of SDEs has been addressed in detail in ([6, 7]). The work has identified the issues related to the existence of low-bandwidth, error-prone links in a multiuser SDE and the need for a new model to address the problems. Intelligent prefetching and caching and the use of *proxy clients* as user agents have been proposed to reduce bandwidth consumption. It makes a valuable contribution in terms of building a complete system within the framework of Oz to solve the various problems. This includes solutions for concurrency control and data reintegration.

The work is motivated by the same set of problems as ours but the approaches have been different. While ([6, 7]) has attempted to provide solutions through experiences with a specific SDE, we have been interested in identifying decisions that need to be made for efficient low-bandwidth operation. and in building a general framework that is rich and flexible enough for use in different SDEs. There are some clear differences. For example, ([6, 7]) treats low-bandwidth and zero-bandwidth operations as being distinct with different solutions in each case. In our model, we have considered zero-bandwidth as a special case of low-bandwidth

operation and have recognized that this allows us to apply techniques like intelligent prefetching in both cases. Again, ([6, 7]) proposes intelligent prefetching of resources prior to detachment (low bandwidth operation) for bandwidth conservation. The prefetching is process-based, i.e. the process bases its prefetching decisions on a user's intent to execute certain process steps while detached. We advance this idea one step further by letting RELOCATE decide which process fragments (if any) a detaching user should be allowed to execute and then downloading the resources for those steps.
The two pieces of work assume the same basic decentralized process model and attempt to address the same set of concerns. Decision- making frameworks and experiences with SDE-specific implementations are both important in building efficient systems. Hence we believe that the two approaches are not divergent and can eventually be combined to efficiently support detached operations in SDEs.

Conclusions and Future Directions

In this work, we have identified the issues raised by the use of mobile computers and wireless technology in PSDEs. We have presented a formal framework for determining whether and how to allow a user to detach from the wired network and work with a low-speed wireless connection to the backbone. Our aim has been to make the framework rich enough to incorporate the high degree of complexity encountered in practical PSDEs and to leave enough flexibility for using it in diverse environments. We have also tried to illustrate some features of the framework by means of a detailed practical example.
To make our problem tractable, we have focussed on the case of a single mobile user. A much higher level of complexity can be expected for the case with N ($N > 1$) mobile users. Hence we can expect modifications to the to the design of the cost functions and to the objective function minimization procedure . In that case, the use of intelligent heuristics will play a significant role in reducing the complexity of the computational process.
We can conceive of subproblems that cannot be solved by this framework in its present form. A number of choices may be available for selecting the set of resources that are needed for a given task, eg. for a collaborative activity, there may be a choice of browsers. One may be sophisticated but may have high processing and storage requirements, the other may be simple but low-overhead. Thus we can possibly have multiple inconvenience costs associated with the execution of a task at any site. The computational framework needs to be enhanced to handle such decisions.
We have limited ourselves to just specifying the cost functions in terms of various factors in order to keep our framework general enough. It will be interesting and challenging to define the exact functions in the context of a specific SDE. In order to gain insights into the functioning of the decision framework and to fine-tune it, it is very important to design and build practical SDEs that use the RELOCATE engine for its relocation decisions.

Acknowledgement

We would like to thank to Prof. Jim Kurose for his support and encouragement at the outset, and for his valuable comments and feedback at various stages of this work.

REFERENCES

[1] G.H. Forman, J. Zahorjan. The Challenges of Mobile Computing. *University of Washington CSE Technical Report* No. 93-11-03.

[2] M.T. Le et al. InfoNet : the Networking infrastructure of InfoPad. *In Proceedings of Compcon*, Calif. Mar. 1995.

[3] B. Schilit and D. Duchamp. Adaptive Remote Paging for Mobile Computers. *Dept. of Computer Science, Columbia University*, technical report TR CUCS-004-91.

[4] J.J. Kistler, M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, Vol. 10, No. 1, February 1992, pp. 3-25.

[5] K. Narayanaswamy, N. Goldman. Lazy Consistency : A Basis for Cooperative Software Development. *In Proc. Computer Supported Collaborative Work* November 1992.

[6] P.D. Skopp, G.E. Kaiser. Disconnected Operation in a Multi-User Software Development Environment. *IEEE Workshop on Advances in Parallel and Distributed Systems* Oct '93 pp. 146-151.

[7] P.D. Skopp. Low Bandwidth Operation in a Multiuser Software Development Environment. *Department of Computer Science, Columbia University*, technical report TR CUCS-035-95.

[8] S.M. Sutton, D. Heimbigner, L.J. Osterweil. APPL/A : A Language for Software-Process Programming. *ACM Trans. on Software Engineering and Methodology*, Vol. 4 No. 3 July 1995, pp. 221-286.

[9] I.Z. Ben-Shaul, G.E. Kaiser, G.T. Heineman. An Architecture for Multi-User Software Development Environments. *Computing Systems, The Journal of the USENIX Association*, 6(2):65-103, University of California Press, Spring 1993.

[10] I.Z. Ben-Shaul, G.E. Kaiser. A Paradigm for Decentralized Process Modelling and its Realization in the Oz Environment. *Proceedings of the Sixteenth International Conference on Software Engineering*, May 1994.