Relocation in Mobile Process-Centered Software Development Environments[*]

Supratik Bhattacharyya
Leon J. Osterweil
Department of Computer Science
University of Massachusetts Amherst
Amherst MA USA
{bhattach,ljo}@cs.umass.edu

Abstract

This paper demonstrates how analysis of a software development process specification can support effective planning for accomodating mobile users of process-centered software development environments. The paper presents a flow graph analysis-based approach to responding to a user who asks to detach from a high-speed wired link, and to continue working through a lower speed interruptible link. We propose the design of an analysis engine that can evaluate the negative consequences of such a request. The purpose of the engine is to evaluate the expected effectiveness of prefetching, caching, and process pruning in mitigating these negative consequences. The engine analyzes a flow graph structure derived from the process specification. The flow graph is annotated with a variety of types of information about the context of the detachment request. The engine takes into account both this context information, and a broad range of other relevant factors, such as speed and reliability of the mobile link, the capabilities of the mobile workstation, the nature and state of the development process and the importance of the detaching user. The paper also describes how predefined heuristics/guidelines can be use to simplify the engine's analysis, by helping it to eliminate obviously poor choices. A detailed example is used to illustrate the workings of this engine.

1 Introduction

This paper presents a conceptual structure and analytic procedures for determining whether, when, and how to allow a member of a team to interact with the rest of the team by means of a wireless communication link that is prone to interruption and degraded quality of service.

The growing availability of wireless communications is changing the ways in which we do many jobs. Especially when coupled with distributed computing approaches, wireless communications make it possible for teams of workers to collaborate in the performance of work, even when some of the team members are travelling, or do not have access to high speed wired communications networks.

While this inviting picture of "anywhere, anytime" collaboration is appealing, it is not without significant drawbacks and dangers. Wireless communications links are invariably slower than wired links, and wired links are far more susceptible to interruptions and transmission errors. As a consequence, it is all too possible for team members connected by wireless communications links to become suddenly and unexpectedly unable to communicate with other members of the team. If this happens at a time when the rest of the team is in urgent need of the services of such a team member, then the productivity of the entire team can be adversely affected.

[*]This research was supported by the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contracts F30602-94-C-0137 and F30602-97-2-0032. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory/IFTD or the U.S. Government.

Because of these possible difficulties it is prudent to exercise caution in allowing a team member to detach from a wired communications link and ”go wireless”. When interactions with such a team member are not going to be required, or are not going to be critically important, for a while then detachment should be allowed. But when close interaction with the team member is expected to be important, it is prudent to not allow that team member to ”go wireless.”

Because the appeal of allowing team members to go wireless is so great, it seems particularly important to find ways that these positive effects are not outweighed the by negative effects of the inopportune loss of contact with a team member. Being able to predict when communication with team members is going to be required or desirable provides a basis for removing or mitigating these negative effects. While it may seem that human intuition should be a good guide in making these decisions, it also seems true that a sound analytic basis for such decisions should be welcomed. We argue that most software development processes need to incorporate responses to a wide variety of contingencies, and that these contingencies can cause process execution to go off in directions that might tax human intuition. There are other cases where the sheer complexity of factors that could contribute to the need for wired communication either numbs or defeats human intuition. In such cases the availability of analytic results should be useful in supporting or guiding decisions that may ultimately have to be made by humans.

In this paper we suggest a way in which it is possible to predict the needs for communication with members of a software development team, and present analytic approaches to using this information as the basis for deciding whether, when, and how to allow team members to go wireless.

The central premise of this approach is to require precise specifications of the work processes and to use those specifications to make firm projections of communications needs. These projections, coupled with knowledge of projected communications characteristics can then provide strongly supported guidance. In this paper we require that the work of the team be represented by a workflow or process specification structure. The paper indicates the features and annotations of this structure that are necessary in order to assure that it can be used to effectively determine whether, when, and how to reply to the request that a team member be allowed to go mobile.

1.1 Scenario

Let us suppose that a team of software designers is engaged in the development of the design of a large software system. Suppose that the requirements for this system have been developed and are available in the form of a large file that must be shared by the various members of the team. Assume that the team members have divided up the design task in such a way that each is responsible for a different part of the design. In general, however, it must be expected that, from time to time, the different parts of the design must be compared and reconciled with each other, the decomposition of the design will have to be reconsidered, and perhaps altered, and some of the design work will suggest the need for changes in the requirements specification.

Thus, while the design activity has been decomposed, it is to be expected that the different designers will need to remain in contact with each other, and will continue to need to share access to both the shared requirements specification and to each others work. Clearly, it is not going to be necessary for all team members to be in constant contact with each other. But it also seems clear that there are likely to be times (eg. when a requirements specification change is being contemplated) when it will be urgently important for all team members to be in contact with each other. The central question, then, becomes how to be sure that team members will not be inaccessible when it is urgently necessary to communicate with them.

In earlier work we have already suggested numerous reasons why it is useful for software development processes such as design to be represented by process programs that have been defined with considerable rigor [9, 13, 14]. Here we observe that, such rigorously defined process programs have additional value in that they can be used to support effective reasoning about the feasibility of detachment requests from team members.

Thus, suppose that the collaborative design process has previously been defined. Suppose that the process definition formalism mandated the specification of the various steps of the process and

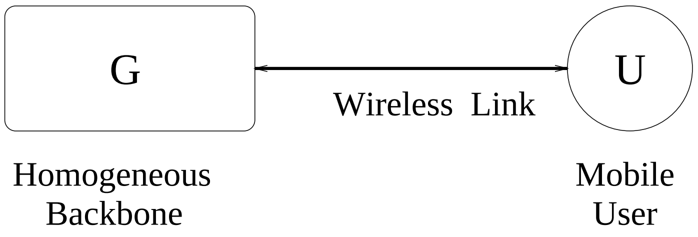


Figure 1: Network topology for single mobile software developer

the data and control relations among the steps. Then, if in addition, each step is also annotated with an estimate of the amount of time the step is expected to take, the team member responsible for completing the step, and with the resources and data needed to carry the step out. Under these circumstances we show how effective reasoning about the impact of a team member going wireless can be computed. In order for this to happen it is also necessary that the progress of execution of the process must be tracked and recorded.

Under this set of assumptions, then let us suppose, for example, that one of the design team members wishes to go wireless. Suppose that the team member indicates that the proposed detachment is to take place in one hour and is to persist for a period of two days (ie. over a weekend). By examining the current status of the execution of the design process it should be possible to determine how reasonable it might be to honor the request. If, for example, the team member has just begun a design step that can be done independently, and that this step is estimated to take two days, then the detachment request should be honored. If, on the other hand, analysis of the current process state indicates that an important team consultation is likely to happen soon (eg. perhaps it is clear that the design team has been going through a period of serious instability and a critical review seem imminently necessary), then the detachment request should be honored only if a certain level of wireless accessibility can be assured. The possibility of any of a number of potential contingencies arising needs to be factored into this decision, and the combination of all possible configurations of contingencies may well make the decision less obvious than might have been thought at first.

In this paper we define a function that is designed to quantify the inconvenience that a requested disconnection is likely to cause. We then demonstrate how evaluation of this function can be the basis for determining whether, when, and how the disconnection request can be honored. We indicate that there are a variety of ways to respond when the proposed disconnection is likely to cause serious inconvenience. Among these responses are: disallowing the disconnection, migrating required resources (if there is sufficient time and capacity to do so), reassigning team members to different tasks, and altering the process itself.

In the next sections we provide details and definitions of the characteristics of the process program formalisms needed to support this scenario. We also indicate how to design functions to measure inconvenience and how to perform the analyses needed to support effective decision making.

2 APPROACH TO THIS WORK

As noted above, the approach we take is to hypothesize a set of models, and to indicate how their analysis can be used to support decisions about disconnection. In this section, we begin by specifying hardware, software, and communications models. We then provide an overview of the architecture of a system, DRAEn, that could support the kinds of analysis we have indicated and we address the question of constructing objective functions whose optimizations represent answers to detachment questions.

2.1 Network model

We assume that communication among software engineers is supported by a mixed wired/wireless communication network. The backbone is a high-speed wired network of arbitrary topology. Users

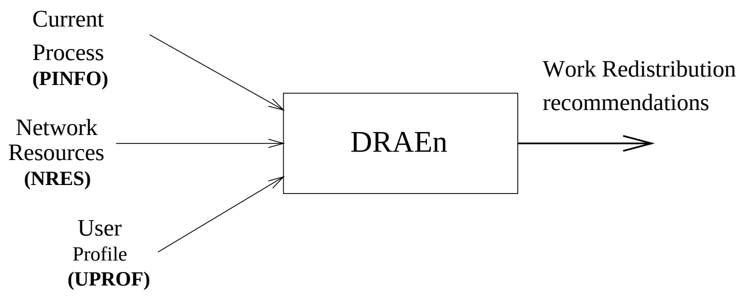


Figure 2: High-level view of DRAEn engine

are connected to the backbone by high-speed links in the attached mode and by low-speed interruptable wireless links in the detached mode.

A simple abstraction of this network configuration is a star topology. The central node of the star represents the entire wired part of the network. All stationary users attach to the center. Software resources can be *physically* located anywhere in the backbone. But because all are interconnected by high-speed links, we abstract their location to a single *logical* site. A mobile user is sited at the end of an arm of the star, connected by an edge representing a wireless link. This simple model encourages a focus on communication over the star's arms and is sufficient for addressing a number of key issues related to user detachment requests. In the rest of the paper, we have focussed on a single mobile developer U who wishes to go mobile, thereby creating a separation from the rest of the development team (located on the homogeneous backbone), collectively referred to as G (Figure 1).

2.2 Software process definition

While the literature contains descriptions of process representations based on Petri Nets [1] , dataflow diagrams [4], or by executable code [8], in this paper we assume that the process is modelled by a graph structure of **tasks** executed procedurally or as triggered reactions to events. We assume that tasks may often be executed in parallel. A task is defined to be an atomic unit of work performed either by one or more **agents**, where an agent is defined as an entity (human or a software program) charged with the responsibility of performing work. In this paper, we consider only two agents - the detaching user U and the rest of the development team G. A task may also contain specifications of artifacts required and produced and of a set of resources required. While one cannot always be sure which tasks will necessarily be executed, and which resources will be used, some of this information is determinable by static analysis, and DRAEn computes and uses this information where possible. DRAEn also uses dynamic monitoring information to analyze the expected consequences of detachment requests.

2.3 DRAEn architecture

Figure 2 is a schematic of our approach, centering on the role of the DRAEn detachment request analysis engine. DRAen accepts three types of input data describing the detachment scenario. A detailed discussion of the various input types follows later in this section. Based on the input, DRAEn constructs an objective function representing user inconvenience, evaluates it for all the ways in which tasks can be assigned to users, and determines the assignment that optimizes the objective function. The output is a recommendation on how to honor the request so as to minimize the objective function measuring inconvenience. This recommendation will be a suggestion on how to reassign tasks among users U and G and which tasks not to execute until the end of the period of detachment. The recommendation will also include the inconvenience cost. If the computed inconvenience exceeds some threshhold, the requested detachment may be deemed inadvisable.

We anticipate that DRAEn will be the core component for a decision-making engine that is rich enough and flexible enough to solve a range of problems of varying degrees of complexity. For example, it is intended to provide “yes/no” answers to relatively simple questions such as whether a detaching user will be allowed to download a single data file. On the other hand, it can be given an entire software process, which it could then use to provide caching, prefetching, process pruning, and work redistribution recommendations. The recommendations are aimed at providing “global benefits” within the entire project. This is supported through the design of the objective function that is used to define and measure inconvenience to all users.

2.4 The objective function

We assume that the goal of DRAEn is to assure that all users are affected as little as possible by detached operation, and it seems reasonable to quantify the discernible adverse impact of detachment as a cost to be mininized by DRAEn. 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. Next, we formulate the inconvenience cost for each factor as a function. The final inconvenience cost function is a weighted sum of these subfunctions and this is minimized by DRAEn in order to find the detachment solution that least adversely affects all users. We recognize that a range of different cost functions may prove useful/appropriate under differing circumstances. We have identified one – user inconvenience – in order to provide focus and specificity for this paper.

All the subfunctions that constitute the inconvenience cost function are defined in terms of the three types of data used to characterize the detachment request.These types of data, which are the inputs to DRAEn, are defined in the next section.

3 INPUTS TO DRAEn

The input to DRAEn consists of three types of information (Figure 2) : PINFO (Process Information), NRES (Network resource availability) and UPROF (Detachment Profile). Each is now described in greater detail.

PINFO is the information about the software process itself. It characterizes the process representation with both static and dynamic information. It is specified as an annotated flowgraph $FG =< N, E >$, where N is a set of nodes, each node representing a task, and E is a set of directed edges, representing all the possible transitions between tasks in N. Therefore, an edge $e(n1, n2) \in E$ implies that task $n2$ can be started only after completion of task $n1$.

Each node $n \in N$ is annotated with the following information :

- **Initial work allocation** (A_n) : For our work, we assume that there exists a pre-allocation of work among the agents U and G prior to the detachment request. Accordingly, each task has a list of execution agents A_n associated with it. Since there are only two agents U and G in our model, A_n can be either $\{U\}$ or $\{G\}$ or $\{U, G\}$.

- **Time of execution at U (X_{nU}) and at G (X_{nG})** : The need to make a distinction between the execution times of the task at the two sites arises due to the possible limitation on the processing power of the mobile computer. These could be simply constant values. However, since it is difficult to predict the exact amount of time that will be required to execute a task, we assume that the probability distributions $F_n^U(t) = P(X_{nU} <= t)$ and $F_n^G(t) = P(X_{nG} <= t)$ are provided. Note that this information may change as the process executes.

- **Value of task** (V_n) : All software tasks may not be not equally important and the value of the task denotes how important it is to execute the task. As will be seen later, this information helps DRAEn to arrive at its recommendations. This information may also change dynamically with time.

- **Resource set** (R_n) : The set of resources $r_1, r_2, \cdots$ that are required for the execution of task n. The dynamic state of the resources, eg. the status of locks on shared resources, values of variables/objects in repositories, etc. must be provided as part of R_n.

Each edge $e(n1, n2),\ n1, n2 \in N$, is annotated with a probability value $P(e(n1, n2))$, representing the probability that task $n2$ will actually be executed immediately after task $n1$ is completed. Note that if task $n1$ is assigned to both U and G, then the edge semantics mandate that both have to complete tasks $n1$ before task $n2$ can be started.

For the current work, the following are the pieces of information that constitute NRES :

- **Available bandwidth,** (B). This is the wireless bandwidth available to the detaching user during the period of detachment.

- **Disconnect Probability,** Q_d. The detaching user may be completely inaccessible at certain times during the period of detachment, either due to the error-prone nature of the wireless connection, or due to the the nature of U's movement. Let us define the disconnect probability, Q_d, as the probability that the user is inaccessible at any time during the detachment period. We also define the **effective bandwidth** for the detached user as $EB = B * Q_d$. EB is one possible measure of the quality of the wireless connection for the detaching user. Other, possibly more sophisticated, measures could be considered if this simple measure proves inadequate or inappropriate. DRAEn computes EB from the supplied values of B and Q_d, and factors it into its analysis, as we shall see in later sections.

- **Storage Capacity of the mobile user's computer**. Mobile computers are designed for portability and low power consumption, and may (but may not) have reduced storage capacity. DRAEn needs to know the storage capacity S_{mobile} and the processing power C_{mobile} of the mobile computer, so that it does not attempt to overload it. Note that a mobile computer may also have low processing power but that is already taken into account in the execution times of different tasks at U and G.

At the time of requesting detachment, a mobile user has to provide DRAEn with certain pieces of information, collectively referred to as UPROF, that aid DRAEn in the decision-making process :

- **Expected period of detachment (**Y_{detach}**)**. DRAEn requires this to predict the process steps that may be executed while the user is detached.

- **User priority**. This is specified as a pair of values (W_U, W_G). When constructing the objective function as a weighted sum of subfunctions, W_U and W_G are used as weights for subfunctions that quantify the inconvenience caused to U and G respectively. This helps DRAEn in reaching its recommendation about task and resource allocations - for example, requests from "important" users may be allowed to inconvenience others signficantly.

- **estimated time before detachment (**Y_{left}**)**. This is useful in determining what can be accomplished (in terms of work redistribution, prefetching, etc.) before the mobile user detaches. This information is not directly used by DRAEn in its current form, but we can anticipate developing a more powerful engine, built around DRAEn, that would use this information in conjunction with the recommendations of DRAEn to arrive at decisions regarding work redistribution, process modifications and resource allocation.

4 STRUCTURE OF THE DRAEn ENGINE

DRAEn reaches its decisions via a procedure depicted by the flow graph in Figure 3. The procedure consists of three stages, of which the second is further divided into two sub-stages.

The first stage of the engine, BUILD, builds a *task precedence graph* (TPG) to represent the precedence relation among tasks in the given process fragment. The ANALYZE module scans

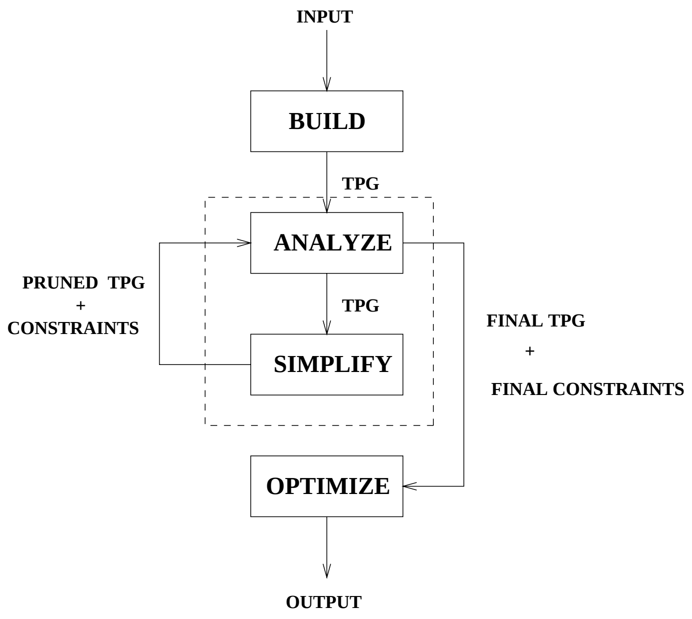


Figure 3: DRAEn engine stages

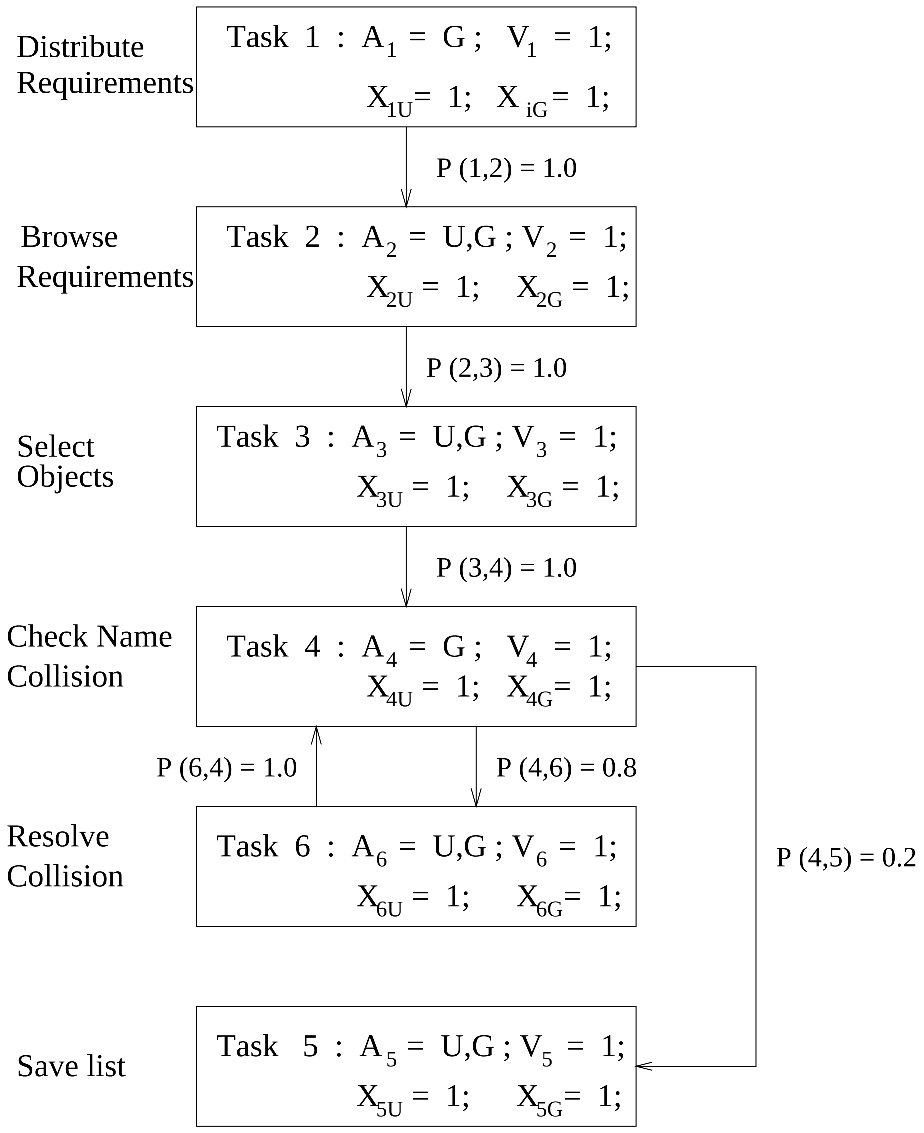
the TPG and evaluates the difficulty or complexity of actually analyzing the set of tasks being presented to DRAEn. If the analysis is deemed “too difficult”, then SIMPLIFY is invoked to perform simplifications. The OPTIMIZE module then weighs the various alternatives and selects the “best” solution, by optimizing an objective function.

Before proceeding further with the description of DRAEn, we introduce a small fragment of a software design process that will be used to illustrate and explain each of the modules of DRAEn. The example is a small fragment of a process for coordinating a team that is carrying out an object-oriented design(OOD) Process. In this example, a team of designers collaborates to identify a set of objects from the requirements specifications provided. The requirement specifications are assumed to be cleanly divided into a number of sub-sections. Each designer is assigned one or more sub-sections 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 any 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.

The software resources involved in this design process fragment are :

- The requirement specifications.
- The list of objects L generated by the designers.
- A browsing tool.
- An object entry tool for adding objects to L.

We illustrate the specification of PINFO for this example as follows. For the sake of simplicity, each task in the flow graph for the OOD (Figure 4) is assigned an importance value of $V_i = 1$ and the time of execution of each is set to $X_{iU} = X_{iG} = 1.0$. The initial allocation of work between U and G is shown. The resource list for each task is omitted for clarity. PINFO also specifies that the probability of iterating one more time around the task 4-task 6 loop is 0.8. This is represented by the annotation (0.8) for edge $(4, 6)$ and the annotation (0.2) for edge $(4, 5)$.



Distribute Requirements
Task 1 : A_1 = G ; V_1 = 1;
X_{1U} = 1; X_{iG} = 1;
P (1,2) = 1.0
Browse Requirements
Task 2 : A_2 = U,G ; V_2 = 1;
X_{2U} = 1; X_{2G} = 1;
P (2,3) = 1.0
Select Objects
Task 3 : A_3 = U,G ; V_3 = 1;
X_{3U} = 1; X_{3G} = 1;
P (3,4) = 1.0
Check Name Collision
Task 4 : A_4 = G ; V_4 = 1;
X_{4U} = 1; X_{4G} = 1;
P (6,4) = 1.0
P (4,6) = 0.8
P (4,5) = 0.2
Resolve Collision
Task 6 : A_6 = U,G ; V_6 = 1;
X_{6U} = 1; X_{6G} = 1;
Save list
Task 5 : A_5 = U,G ; V_5 = 1;
X_{5U} = 1; X_{5G} = 1;

Figure 4: Flow graph for two-user collaboration on a design process fragment

DRAEn has to analyze the detachment request and arrive at task allocation decisions. In the following sections, we will demonstrate what each stage of DRAEn does to reach the recommendation of which tasks to assign to which user.

4.1 The BUILD module

The BUILD module builds a Task Precedence Graph (TPG) from the process flow graph. A task precedence graph is a directed acyclic graph whose nodes are derived from those of the flow graph and satisfy the condition that every TPG node must be allocated to only one agent. This, if task n in the flow graph is initially allocated to both U and G, then two nodes are created in the TPG to represent this collaboration. Moreover, all flowgraph loops are unrolled a (finite) number of times in the TPG to anticipate all iterations that are expected during the proposed period of detachment. If a flowgraph node N has more than one child task, then the TPG representation shows that none of them can be started before N is completed. Likewise, if a flowgraph node N has multiple incoming edges, then all the predecessor tasks have to be completed before N can be started.

Each edge of the TPG is annotated with a probability value derived from the edge probabilities in the flow graph. The exact way in which this is accomplished will be illustrated later through an example. Each node is annotated with all the information corresponding to the node in the flow graph that it is derived from, except that the initial allocation list A_n for any node n in the TPG has either U or G, not both. In addition, each node n is annotated with a value P_n indicating the probability that the task will *actually* execute during the detachment period. Let us now consider how this probability value is computed. Let the nodes in the TPG be numbered $1, 2, \cdots, M$ and $f(i, j)$ be the directed edge from node i to node j. Let $P(f)$ be the probability associated with edge f in the TPG, where $f \in F$, the set of edges in the TPG. Let S_n and E_n be the starting time and ending time of task n and X_n be its time of execution. Note that all time values are measured relative to the beginning of the period of detachment. Then we have

$$P(E_n <= Y) = \int_{t=0}^{Y} P(S_n \leq (Y - t)) * P(X_n \leq t) dt \tag{1}$$

Let us associate a **predecessor set** $PRED_n$ with each task n, defined as

$$PRED_n = \{j : j \in N \; and \; f(j, n) \in F\} \tag{2}$$

i.e.$PRED_n$ is the set of tasks from which there is an edge into task n.

Therefore

$$P(S_n \leq Y) = \Pi_{j \in PRED_n} P(E_j \leq Y) * P(f(j, n)) \tag{3}$$

The first term of the right hand side of equation 3 represents the probability that a predecessor $j \in PRED_n$ finishes executing before the end of the detachment period. The second term, provided as part of PINFO, represents the probability that task n will be executed at all, even if task j is completed.

Then P_n is given by

$$P_n = P(S_n \leq Y_{detach})$$

Thus we see that P_n is determined by the TPG edge probabilies, the estimated period of detachment and the time of execution of the predecessors of task n. Finally, a weight $W_n = P_n * V_n$ is associated with each task n in the TPG, where V_n is the value of task n. We shall see later that DRAEn uses W_n to weigh the relative importance of task n while making decisions regarding task/resource allocations.

4.2 Algorithm FG-To-TPG

This section describes the algorithm for constructing a task precedence graph (TPG) from a flow graph (FG). As a simplification, the algorithm assumes that there are two agents U and G. A generalization to account for an arbitrary number of agents seems straightforward.

Inputs :

1. **FG_Nodes** : The set of N nodes, $n_1, n_2, \ldots, n_N$ in the input FG. The information with which each node is annotated has been described earlier.

2. **FG_Edges** : The set of edges in FG where each edge is represented by $e_1(n_{11}, n_{12}), \cdots, e_M(n_{M1}, n_{M2})$.

3. **H** : The maximum height to which the TPG is to be built. Note that loops in the FG are unrolled in the corresponding TPG, hence the height of the TPG can be arbitrarily large. Thus it is useful to specify an upper bound on the height of the TPG that we want to consider for the subsequent analysis.

Outputs :

1. **TPG_Nodes** : The set of TPG nodes created from the nodes of FG. Each TPG node is denoted as $N_n^{a,h}$, where n is the node of the FG from which it is created, a is the identity of the agent that is assigned this task and h is the level in the TPG at which this node appears. The root node of the TPG is considered to be at level 0 and a node at level i has children at level $i+1$.

2. **TPG_Edges** : The set of edges connecting the nodes in the TPG, $F = (N_{n_i}^{a,h}, N_{n_j}^{b,h+1}) : N_{n_i}^{a,h}, N_{n_j}^{b,h+1} \in TPG\_Nodes$, where the agents a and b may be same or different, and $(n_i, n_j) \in E$, the edge set of the flow-graph, FG.

Data Structures :

1. **WL** : A list of FG nodes. This list is used to determine which node of the FG is to be processed to create TPG nodes. When an FG node is inserted into WL, it is tagged with its h value – each TPG node created from this FG node will appear at height h in the TPG.

2. **UL** : A list of FG edges. This list is used for creating TPG edges from already existing TPG nodes to newly created ones.

Functions : For ease of exposition, we will assume that the following set of functions is provided to us :

1. **FG_Root(FG)** : returns the root node of a given FG.

2. **FG_Child(n)** : returns the set of child nodes of node n in a given FG.

3. **FG_OutEdges(n)** : returns the set of outgoing edges from node n in a given FG.

4. **FGNode_Get_Agents(n)** : returns the set of agents that are initially allocated the task corresponding to node n in an FG. This list can take one of three possible values : $\{U\}$,{G}or,{U,G}.

5. **TPG_Create_Node(n,a,h)** : creates a new TPG node $N_n^{a,h}$. This node is annotated with all the information corresponding to n, *except* for the agent a, which in this case, is explicitly specified. In addition, it is annotated with its level h in the TPG.

6. **TPG_Create_Edge(N1,N2)** : creates a new TPG edge from node $N1$ to node $N2$.

7. **WL_Is_Empty(WL)** : returns TRUE if WL is empty, FALSE otherwise.

8. **WL_Retrieve(n,h)** : returns the next FG node n and the associated level tag h from WL and removes the entry from WL.

9. **UL_Remove ((n_i, n_j))** : removeS edge n_i, n_j from UL.

Function **Build_TPG (FG_Nodes,FG_Edges,H)** :

Initialization :

$TPG\_Nodes \leftarrow \{\}$.
$TPG\_Edges \leftarrow \{\}$.
$WL \leftarrow \{Root(FG), 0\}$.
$UL \leftarrow \{\}$.

```
While (not WL_Is_Empty(WL))
        WL_Retrieve($n, h$).
        Agentset = FGNode_Get_Agents($n$).
        Case (AgentList)
        $\{U\}$ : Build_Nodes_And_Edges_1($n, U, h$);
        $\{G\}$ : Build_Nodes_And_Edges_1($n, G, h$);
        $\{U, G\}$ : Build_Nodes_And_Edges_2($n, h$);
        If $(h < H)$
          For each $m \in$ FG_Child($n$)
              WL $\leftarrow$ WL $\bigcup \{\{m, h+1\}\}$.
          For each $(n, n^{'}) \in$ FG_OutEdges($n$)
              UL $\leftarrow$ UL $\bigcup \{(n, n^{'})\}$.
```

Function **Build_Nodes_And_Edges_1(**n, a, h**)** :

```
$N_n^{a,h}$ = TPG_Create_Node($n, a, h$).
TPG_Nodes $\leftarrow$ TPG_Nodes $\bigcup \{N_f^{a,h}\}$.
For each $(m, m^{'}) \in$ UL
    If $(n == m^{'})$ then
        If $N_n^{a,(h-1)} \in$ TPG_Nodes then
            NewEdge = TPG_Create_Edge($N_m^{a,(h-1)}, N_n^{a,h}$).
            TPG_Edges $\leftarrow$ TPG_Edges $\bigcup$ {NewEdge}.
        UL $\leftarrow$ UL - $\{(m, m^{'})\}$.
```

Function **Build_Nodes_And_Edges_2(**f, h**)** :

```
$N_n^{U,h}$ = TPG_Create_Node($n, U, h$).
TPG_Nodes $\leftarrow$ TPG_Nodes $\bigcup \{N_n^{U,h}\}$.
$N_f^{G,h}$ = TPG_Create_Node($n, G, h$).
TPG_Nodes $\leftarrow$ TPG_Nodes $\bigcup \{N_n^{G,h}\}$.
For each $(m, m^{'}) \in$ UL
    If $(n == m^{'})$ then
        If $N_m^{U,(h-1)} \in$ TPG_Nodes and
        $N_m^{G,(h-1)} \in$ TPG_Nodes then
            NewEdge = TPG_Create_Edge($N_m^{U,(h-1)}, N_n^{U,h}$).
            TPG_Edges $\leftarrow$ TPG_Edges $\bigcup$ {NewEdge}.
            NewEdge = TPG_Create_Edge($N_m^{G,(h-1)}, N_n^{G,h}$).
            TPG_Edges $\leftarrow$ TPG_Edges $\bigcup$ {NewEdge}.
        else
            If $N_m^{U,(h-1)} \in$ TPG_Nodes then
                NewEdge = TPG_Create_Edge($N_m^{U,(h-1)}, N_n^{U,h}$).
                TPG_Edges $\leftarrow$ TPG_Edges $\bigcup$ {NewEdge}.
                NewEdge = TPG_Create_Edge($N_m^{U,(h-1)}, N_n^{G,h}$).
                TPG_Edges $\leftarrow$ TPG_Edges $\bigcup$ {NewEdge}.
            If $N_m^{G,(h-1)} \in$ TPG_Nodes then
                NewEdge = TPG_Create_Edge($N_m^{G,(h-1)}, N_n^{U,h}$).
                TPG_Edges $\leftarrow$ TPG_Edges $\bigcup$ {NewEdge}.
                NewEdge = TPG_Create_Edge($N_m^{G,(h-1)}, N_n^{G,h}$).
                TPG_Edges $\leftarrow$ TPG_Edges $\bigcup$ {NewEdge}.
    UL $\leftarrow$ UL - $\{(m, m^{'})\}$.
```

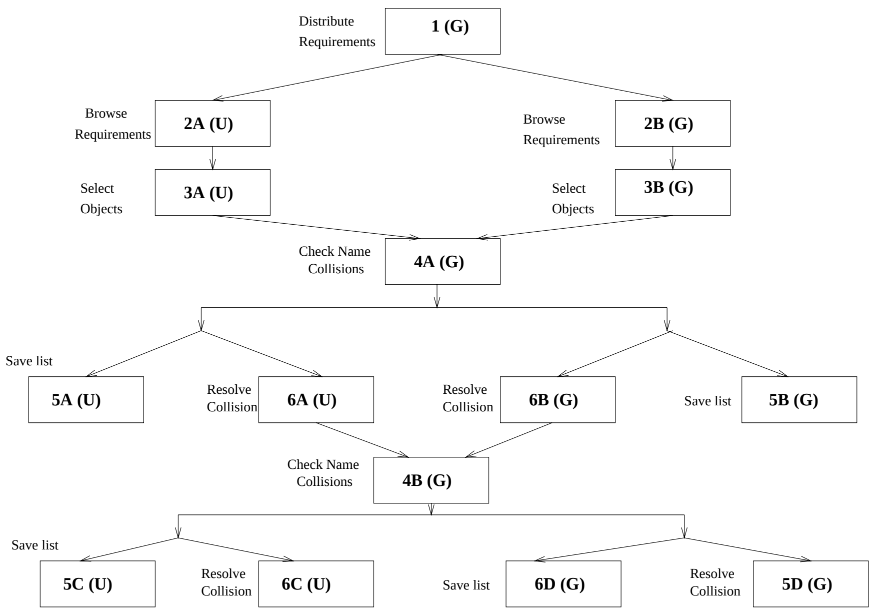


Figure 5: Task precedence graph for design process fragment

4.2.1 TPG for BOOD process fragment

Let us consider the construction of a TPG for the process fragment presented in Section 4. Figure 5 represents the TPG constructed by the BUILD module from the flow graph for this process fragment (Figure 4). For the sake of clarity, the node and the edge annotations have been omitted from the figure. Assume that the requirement specifications are divided into two parts and one is assigned to each of U and G. Tasks 2 and 3 in Figure 4 represent the activities of U and G browsing through their respective specifications and selecting objects. Hence in Figure 5,each is represented by two tasks - $2A$ and $3A$, representing U's activities and $2B$ and $3B$, representing G's activities. Objects selected by U and G after these steps are compiled in two separate lists. The name collision check across these two lists is to be performed only by G (task $4A$). Once this check is completed, one of two cases can arise :

- No collisions are found and U and G write out their respective lists to a repository (tasks $5A$ and $5B$).
- Collisions are found. In that case, U and G have to make modifications to their lists (tasks $6A$ and $6B$) to resolve the collisions. After that, G has to perform the name collision check once more (task $4B$).

Only two iterations of this check-resolve-save loop are represented in Figure 5, since the probabiliies specified indicate that the probability of more than two iterations is negligibly small. The complete list of edge probabilities for the TPG, derived from the edge probabilities and loop iteration probabilities of the OO Design flow graph (Figure 5), is shown in Table 1.

Knowing the the probability distribution of the execution time of each task and the expected period of detachment, we can now compute the probability value associated with *each node* in the TPG, following equations (1), (2) and (3). For example, if we assume that the detachment period is sufficiently long for all tasks to finish execution, then from Table 1, $P_{5C} = P_{5D} = 0.16$.

Let us close this section by informally discussing some of the possible alternatives that the DRAEn engine will have to consider when processing a detachment request for the BOOD process

Edges	Prob.
$(1, 2A)$, $(1, 2B)$, $(2A, 3A)$, $(2B, 3B)$, $(3A, 4A)$, $(3B, 4A)$	1.0
$(4A, 5A)$, $(4A, 5B)$	0.2
$(4A, 6A)$, $(4A, 6B)$	0.8
$(6A, 4B)$, $(6B, 4B)$	1.0
$(4B, 5C)$, $(4B, 5D)$	0.8
$(4B, 6C)$, $(4A, 6B)$	0.2

Table 1: Edge probabilities for BOOD TPG in Figure 5

fragment example. In the next section, we shall describe a framework for comparing the costs for the various alternatives and choose the one with the minimum cost :

Option 1 : Allow U to take away the process fragments marked with U in Figure 5. This will require the following resources to be downloaded to U :

1. The *requirements specifications list* to be used by U for selecting objects.
2. A *browsing tool* for browsing requirement specifications.
3. An *object entry tool* for compiling a list of objects.

In addition, the compiled list of objects will be uploaded from U to G for name collision check. Some of the factors contributing to the inconvenience cost for this option are :

- Storage constraints on the mobile computer at U.
- Difficulties in uploading the compiled object list to G and in sending feedback to U in case collisions are detected by G.
- The inconvenience of running the browser and object entry tools on a (possibly) less powerful mobile computer at U.

Option 2 : Reassign all (or part) of U's work to G. This solves some of the problems in Option 1, but the additional load inconveniences G and the cost of that inconvenience has to be considered now.

Option 3 : Not execute any of the tasks assigned to U till U reattaches. This will cause inconvenience to U and prevent the creation of list L. It will also hold up the collision check and resolution activities at G. The total inconvenience cost is the sum total of the inconveniences caused to U and G due to the delay in execution of the postponed tasks.

4.3 The OPTIMIZE module

The purpose of the OPTIMIZE module is to determine the optimal assignment of users to each of the tasks to be executed during the planned period of detachment. Note that the rationale for creation of the TPG was to create a graph each of whose nodes represents a task to be accomplished by exactly one user. The initial annotation of the nodes of the TPG is borrowed from the initial FG submitted to DRAEn as part of PINFO. OPTIMIZE takes the TPG, so annotated, as input, and performs an optimization analysis whose purpose is to determine the optimal reassignment of tasks to users, where optimality is defined with respect to an objective function that quantifies user inconvenience, using other parameters that are taken as input to DRAEn.

As we shall see, determining the optimal assignment of tasks to users has a worst case complexity bound that is cubic in the number of tasks to be assigned. We anticipate that, in many cases, the number of tasks will be modest and it will be possible to determine the optimal assignment by exhaustive computation of all possible assignments. In cases where this is impractical, other optimization techniques may be applicable. Discussion of alternative optimization approaches is beyond the scope of this paper.

On the other hand, the determination of the optimal assignment for this problem seems to be amenable to various sorts of heuristic approaches to simplification. These are discussed in this section. We begin by introducing the objective function whose optimization is the focus of the OPTIMIZE module.

Consider a task precedence graph with N nodes. Let us associate a variable x_i with node i,($i = 1, \cdots, N$), 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 by G.
= 2, if the task is executed by U.

Let us define a class of functions $\{Z_i\}_{i=1}^{N}; X_i : \{0,1,2\} \rightarrow R_+^2$, where R_+^2 is the set of positive real numbers. $Z_i(0,1,2)$ represents the total inconvenience cost for task i when for each of the three execution agent possibilities. Computation of F_i depends on the cost factors that we choose to consider, eg. processing cost, storage cost, cost of reassigning U's work to G, penalty of non-execution of task i, etc. Our goal is to choose the vector $X = (x_1, x_2, \cdots, x_N)$ so as to

$$\text{minimize} \qquad \sum_{n=1}^{N} F_i(x_i) \tag{4}$$

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

- *Precedence relations among tasks in the graph*, eg., if task i precedes task 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 j executes even though its predecessor is abandoned), are not feasible. In order to ensure that no such vector is found to minimize the above objective function, the optimization problem must be constrained. The constraint $2x_i - x_j \geq 0$ suffices to assure the desired results.

- *Process-imposed requirements*. As an example of this, consider the case where the input to DRAEn mandates that tasks i and j must always be executed at the same location. This might be an optional part of $UPROF$. Then any feasible solution has to satisfy the constraint $x_i - x_j = 0$.

- *SIMPLIFY module constraints*. In the next section, we will see that such factors as storage limitation may prevent the assignment of a task to the detaching user. This can be simply 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 computational overhead of the optimization problem.

Solving the above optimization problem, subject to the constraints, will yield a feasible optimal vector $X^* = (x_1^*, \cdots, x_N^*)$, and consequently, an optimal assignment of tasks to users. This will determine which process fragments should be executed at the remote site during the detachment period. Accordingly, resources required for these should be downloaded to U before detachment.

Of course, whether all of them can be downloaded depends on the time remaining before detachment, the capacity of the mobile computer and the capacity of the communication link. If it is not possible to download all resources, some arbitration policy may be used. For example, priorities may be assigned to the resources that will ensure the execution of certain key tasks. The process of determining what to download and how to do so is beyond the scope of this paper.

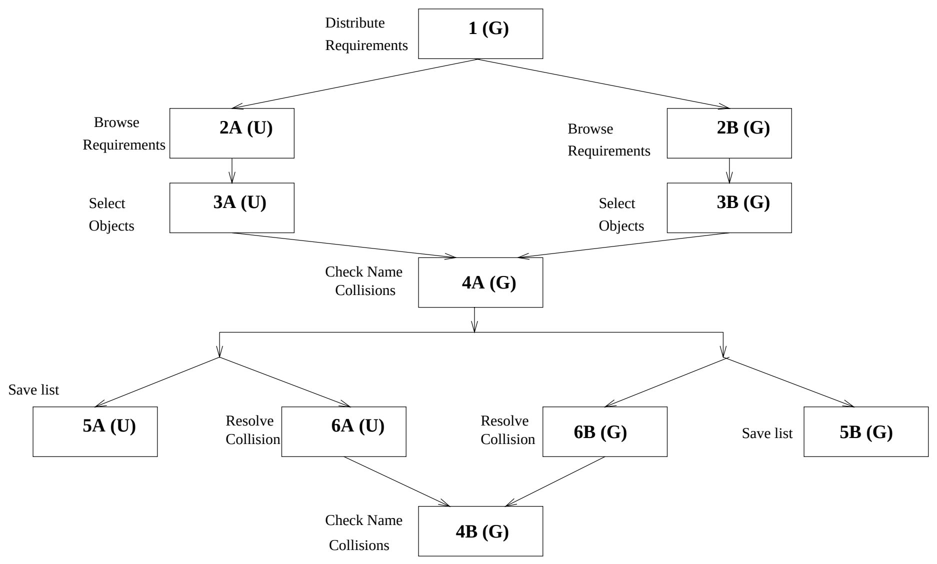


Figure 6: Reduced TPG for design process fragment analyzed by the OPTIMIZE module

4.4 The ANALYZE-SIMPLIFY module

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

1. If the weight W_i associated with task i in a TPG is less than a certain threshold, prune the subtree rooted at that node.
2. Exclude the mobile user from some or all of the collaborative activities during the detachment period. A number of factors have to be considered when making such a decision, eg. the importance of the detaching user for the activity in question, effective bandwidth, etc.
3. If the user has a preference about activities during detachment then honor those preferences.

Note that pruning the TPG (as in example 1 above) simplifies the optimization problem by reducing the number of tasks to be considered by the OPTIMIZE module, and hence, the size of the vector space which has to be searched for the solution to the optimization problem. On the other hand, restrictions such as the ones in examples 2 and 3 can be easily modelled as constraints for the optimization problem and hence can reduce the number of feasible solutions to be considered in the optimization step.

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 problem increases with the number of tasks in the TPG. ANALYZE may repeatedly invoke SIMPLIFY until the number of tasks is sufficiently reduced. Again, if storage capcity was the overriding factor in deciding on task/resource allocations, then ANALYZE may simply invoke SIMPLIFY to set various other costs, eg. processing cost, network bandwidth cost, etc. to zero, so as to ensure that the solution to the optimization problem is not influenced by these factors. As an example of such policy/heuristic-based simplifications, let us consider the TPG in Figure 5. If we prune every task i for which $W_i <= 0.2$, then we obtain the pruned graph in Figure 6, by pruning tasks $5C$, $5D$, $6C$ and $6D$. This is the graph we consider now to complete our example.

4.5 Optimization Example

Let us now apply the optimization technique to the pruned TPG of Figure 6. The precedence relations among the tasks gives rise to a large number of constraints. Since these are straightforward, we do not list them here. However, let us assume that there are certain environment-imposed restrictions, specified as guidelines in the ANALYZE-SIMPLIFY module, and see how we can design constraints to model them :

1. tasks $1, 2B, 3B, 4A, 4B, 5B, 6B$, if executed at all during the detachment period,have to be executed at G. This implies that $x_j \leq 1, j = 1, 2B, 3B, 4A, 4B, 5B, 6B$.

2. All of the tasks $2A, 3A, 5A, 6A$ have to be executed at the same site (U or G) or have to be postponed till U reattaches, i.e. $x_{2A} = x_{3A} = x_{5A} = x_{6A}$.

Thus we have heavily restricted the relocation options. The decision that remains to be made is whether to allow tasks $2A, 3A, 5A$ and $6A$ to execute at one of the sites during detachment or whether to postpone them altogether. Such a postponment will automatically imply the postponment of tasks $4A, 4B, 5B$ and $6B$ as well. Since there is no reason to postpone the execution of tasks $1, 2B$ and $3B$, we set $x_1 = x_{2B} = x_{3B} = 1$. For the remaining variables, there are only three feasible assignments of value sets :

1. $x_{2A} = x_{3A} = x_{5A} = x_{6A} = 0;\ x_{4A} = x_{4B} = x_{5B} = x_{6B} = 0.$
2. $x_{2A} = x_{3A} = x_{5A} = x_{6A} = 1;\ x_{4A} = x_{4B} = x_{5B} = x_{6B} = 1.$
3. $x_{2A} = x_{3A} = x_{5A} = x_{6A} = 2;\ x_{4A} = x_{4B} = x_{5B} = x_{6B} = 1.$

It easy to see how various cost factors influence the optimal choice. For example, if the cost of the inconvenience caused to G due to the reassigment of U's work to G is low, the option 2 will be the optimal choice. On the other hand, if the inconvenience cost of communication/resource transfer over the wireless channel is relatively low and the inconvenience costs of storage and processing power limitations of the mobile computer are negligible, then option 3 may be the better choice. In that case, DRAEn will recommend that U be allowed to take away the process fragment represented by tasks $2A, 3A, 5A, 6A$. Resources like the browser and object entry tools will then have to be downloaded to U prior to detachment.

5 RELATED WORK

The problem addressed in this paper bears some similarity to a class of program optimization problems aimed at increasing throughput in parallel hardware architectures [8]. For example, caching approaches are routinely used to assure that operands needed by instructions in a pipelined processor are readily accessible, thereby avoiding the need to risk long waits (inconvenience) that may be entailed in a fetch from main storage. Prefetching approaches are also used to improve throughput through pipelined and similar hardware architectures.

The similarity of our work to these problems is no accident. In optimizing the utilization of a pipeline, analysis of the tasks to be done (ie. instructions to be decoded and executed) is used to provide hints on how to position and utilize resources for minimized delay (inconvenience). The problem we address in this paper is similar in that we are seeking to minimize the inconvenience caused when an execution device (in our case a user) is going to be handicapped or inconvenienced by the relative distance of needed resources. In our case, the availability of an explicit process representation makes it possible for us to use analysis approaches that bear some similarity to those used in pipeline optimization, in considering similar remedies, namely caching, prefetching, etc.

This work also bears important similarities to other work in mobile computing research. 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 [5]. 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([7, 10]). 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 [6] 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.

The issue of disconnected and low-bandwidth operation in the specific context of SDEs has been addressed in detail in [12, 11]. 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 [12, 11] 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 the core component for a general framework that is rich and flexible enough for use in different SDEs. There are some clear differences. For example, [12, 11] 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, [12, 11] 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 DRAEn arrive at a set of recommendations regarding which process fragments (if any) a detaching user should be allowed to execute and then using these recommendations to decide which resources to prefetch. 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.

6 CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we have presented an approach to dealing with one of the difficult the issues raised by the use of mobile computers and wireless technology in collaborative software development. Users wishing to go mobile can potentially cause serious inconvenience to all users. In a software project, the presence of a process specification enables the quantification of this inconvenience, and supports reasoning about the effectiveness of various remedies. We have presented an approach to determining how to minimize the inconvenience caused by a detaching user. Our aim has been to make the approach rich enough to incorporate the high degree of complexity encountered in practical software development scenario and to leave enough flexibility for using it in diverse environments.

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 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.

Various complicating factors may reduce the applicability of this approach 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 current DRAEn design would need to be enhanced to handle such decisions.

Finally, it is important to note that what we have presented is a design of DRAEn, not an actual implementation. We believe that the design, and the framework for analysis within which it works, should be effective in dealing with a class a problems that will be increasingly troublesome as mobile computing becomes more prevalent. In order to be effective, however, cost functions will have to be specified in detail, and the heuristic approaches presented here will have to be made more precise. Implementations that incorporate such details will be the basis for definitive evaluation of the approach presented here.

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] Bandinelli, S., Fuggetta, A., and Grigolli, S. Process Modelling in-the-large with SLANG". In *Proc. of the Seconf Internation Conference on the Software Process* (1993), pp. 75–83.

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

[3] Ben-Shaul, I., Kaiser, G., and Heineman, G. An Architecture for Multi-User Software Development Environments. *Computing Systems, The Journal of USENIX Association 6*, 2 (Spring 1993).

[4] Fernström, C. PROCESS WEAVER: Adding Process Support to UNIX. In *Proc. of the Seconf Internation Conference on the Software Process* (1993), pp. 12–26.

[5] Forman, G., and Zahorjan, J. The Challenges of Mobile Computing. Tech. Rep. 93-11-03, University of Washington, CSE, 1993.

[6] Kistler, J., and Satyanaranan, M. Disconnected operation in the coda file system. *ACM Trans. on Computer Systems 10*, 1 (February 1992), 3–25.

[7] Lee, M., and al, e. Infonet l the networking infrastructure of infopad. In *Proceedings of Compcon, California, USA* (1995), pp. 779–784.

[8] Muchnick, S. S. *Advanced Compiler Design Implementation*. Morgan Kaufmann Publishers, San Francisco, California USA, 1997.

[9] Osterweil, L. Software processes are software too. In *Proc. 9th International Conf. on Software Engineering, Monterey, CA, USA. (Invited Keynote)* (March 1987), pp. 2–13.

[10] Schilit, B., and Duchamp, D. Adaptive Remote Paging of Mobile Computers. Tech. Rep. CUCS-004-91, Columbia University, Computer Science, 1991.

[11] SKOPP, P. Low-Bandwidth Operation in a Multi-user Software Development Environment. Tech. Rep. CUCS-0035-95, Columbia University, Computer Science, 1993.

[12] SKOPP, P., AND KAISER, G. Disconnected Operation in a Multi-User Software Development Environment. In *IEEE Workshop on Advances in Parallel and Distributed Computing* (1993), pp. 146–151.

[13] SONG, X., AND OSTERWEIL, L. Experience with an approach to comparing software design methodologies. *IEEE Transactions on Software Engineering* (May 1994), 364–384.

[14] SUTTON, S., HEIMBIGNER, D., AND OSTERWEIL, L. APPL/A : A Language for Software-Process Programming. *ACM Transactions on Software Engineering and Methodology 4*, 3 (July 1995), 221–286.