Effective Task Allocation in Distributed Environments: A Traceability Perspective

Salma Imtiaz, Naveed Ikram  
Department of Software Engineering  
International Islamic University  
Islamabad, Pakistan  
salma.imtiaz@iiu.edu.pk, naveed.ikram@bcs.org

Abstract—Task allocation or work assignment in Distributed Environments is a challenging task due to intricate dependencies between distributed sites and fundamental requirement of multifarious information. Conway's law relates product architecture to communication and coordination needs of the people, whereas Parnas argues that communication and coordination needs give rise to technical dependencies. Product structure is depicted in its architecture, which in turn, consists of multiple views based on different perspectives. These views which are used to model different concerns of various stakeholders are inter-related. Task allocation depends on information about different architectural views and their interrelationship. Traceability links between various views can be used to model this interrelationship. There is a need to identify the traceability support between different architectural views to determine the extent of linkage between them. Task allocation is also dependent on factors not depicted in product architecture such as temporal and cultural dependencies between distributed sites. These dependencies highlight the need of an effective and sound task allocation strategy for distributed environment. Task allocation is also important for co-located development, but it acquires a critical value in a distributed setting. Distributed teams need to intercommunicate and coordinate their activities to understand each other’s culture, norms, organizational structures and business process etc., while co-located teams share common social and cultural norms and have almost the same knowledge level. Lack of inter-team information, problem of mapping the system architecture to organizational structure, and time pressure are some of the important factors aggravating the complexities of a distributed environment. 

While considering a mechanism to bridge the gap caused by geographic and cultural barriers in distributed development, we find that architecture plays an important role in this regard. It acts as a central knowledge and coordination mechanism. However, the architecture of a system facilitates identification of some but not all the dependencies. For example, temporal, cultural and knowledge dependencies, which are also critical for an effective task allocation strategy, are not visible in architecture. 

The architecture view type literature highlight allocation view type as necessary to model ‘allocated to’ relationships. This view type is necessary for task allocation as it presents the allocated to relationship between software elements and environmental elements. The environmental elements in case of work assignment style are individuals, teams, and organizational units etc. Thus it focuses on task allocation to teams. The software elements in work assignment view type are elements from the module and the component and connector view type, thus implicitly creating a linkage between various views. Because of this conceptual linkage, we can establish traceability relationships between views for supporting alignment between them. The architectural models surveyed and presented in Section II-A do not explicitly model this view resulting in lack of foundational information for task allocation.

Keywords—Task Allocation; Architectural View; Distributed Development.

I. INTRODUCTION

Allocation of task to distributed teams is a complicated and difficult affair as it involves enlarged time and space dimensions while adequate information of distributed sites is lacking. Current literature adequately identifies the temporal, cultural, knowledge base, communication, coordination and other dependencies, which combined with various other factors, make the task allocation problematic. Currently, task allocation is mostly done with focus on module or component dependencies overlooking most of the above mentioned important factors. This results in inadequate task allocation.

The architecture view type literature highlight allocation view type as necessary to model ‘allocated to’ relationships. This view type is necessary for task allocation as it presents the allocated to relationship between software elements and environmental elements. The environmental elements in case of work assignment style are individuals, teams, and organizational units etc. Thus it focuses on task allocation to teams. The software elements in work assignment view type are elements from the module and the component and connector view type, thus implicitly creating a linkage between various views. Because of this conceptual linkage, we can establish traceability relationships between views for supporting alignment between them. The architectural models surveyed and presented in Section II-A do not explicitly model this view resulting in lack of foundational information for task allocation.
II. MOTIVATION

According to Conway’s law [8], the design of a system reflects the communication and coordination needs of the people. As opposed to this law, Parnas [9] argues that technical dependencies between modules give birth to communication and coordination needs. Both these statements have been validated through empirical evidence and this inter-relationship highlights the need for a clear, effective and sound strategy for task allocation in distributed environment. Considering that both these laws are true can we identify this information as early as required for effective task allocation?. Current literature points to a glaring misalignment between communication/coordination dependencies and technical dependencies; particularly, in distributed environment [10][11]. The current task allocation literature also does not encompass all the factors necessary for effective work assignment in a distributed setting.

An architecture is divided into multiple views for separating stakeholders’ concerns. Modeling each concern in a separate view increases its comprehensibility, reuse and evolution [12]. Where these views are separated for understanding, proper linkage between them is also required for task allocation, evolution and view synchronization [13]. This is where traceability information comes in. This information is used to link the work assignment view with implementation view and execution view etc. to understand the effect of component and runtime dependencies on work assignment [6]. The traceability information between architectural views needs to be correct and current at all times to ensure architecture’s inter-view alignment. We need traceability information within different views to ensure their synchronization in a manner that modification in one view automatically modifies similar information in other views as well. This synchronization is particularly important for re-allocation of work.

Different architectural views have been proposed by researchers and institutes for effective modeling of software architecture. Due to the unique nature of software, the scope of this work only includes architectural models specific to software systems. We have excluded architectural models such as Telemanagement Forum Views whose focus is telecommunication systems [34], Open Group Architecture Framework whose focus is enterprise architecture [35] and Zachmann’s Framework, which again focuses on enterprise architecture [36]. Out of all the architectural models only five identify the need for separation of stakeholders concerns which is necessary for increased understanding, reuse and evolution of architecture. These are: SEI View Model [12], Siemens 4 View Model [12][14][15], 4+1 View Model [12][15][18], Rational ADS View Model [12] and RM-ODP [12] [14][15]. We are interested in architectural viewpoint models which reflect different concerns separately, provide a linkage mechanism between them and focus on design of the system. SEI View Model and ISO-RM do not meet our requirement because of their independent views. Besides the focus of ISO-RM is ‘development across variant domains’. The focus of Rational ADS View Model is ‘requirement evolution’, which is not relevant to task allocation. The only two viewpoint models which focus on architectural design are Siemens 4 View Model and Kruchten’s 4+1 View Model. We have selected Kruchten’s 4+1 View Model because it comprehensively describes the architecture of a system [12]. Different views of this model are designed using UML (Unified Modeling Language) which is an industry standard and a standard way to represent product architecture. It facilitates easy comprehension of different views [16]. Traceability support between these architectural views can be identified by studying the traceability support between UML models present in each view.

Current literature on architecture highlights the need for different views [18]. 4+1 Architectural View Model proposed by Philippe Kruchten is one such model which reflects concerns in different views [18]. It organizes the architecture using five concurrent views namely: logical view, process view, development view, use case view and physical view. All the surveyed architectural view models including 4+1 View Model lack work assignment view which is necessary for task allocation in distributed environment. The full support for this view needs to be incorporated for resolving task allocation problem in distributed development. Both, 4+1 View Model and Rational ADS View Model (extension of 4+1) consist of the deployment view which falls in the category of allocation view type. This view type is restricted to deployment on physical nodes only where deployment of work to different organizational units, teams or individuals is not modeled. Depicting work assignment view via module view is also considered a viable approach but it is also fraught with problems [19].

Task allocation is dependent upon communication and coordination needs of an organization (Conway’s law) and various other factors discussed in literature survey. We present a résumé of the current literature in the following paragraphs. We have divided Section III (Literature Survey) into three subsections. Section A identifies different types of important dependencies existing between various distributed sites and their importance as related to task allocation. Subsection B highlights the current task allocation strategies used in distributed environment and their limitations. Subsection C identifies the traceability support present between architectural views of 4+1 Architectural View Model. Discussion is presented in Section IV whereas conclusion and future work are presented in Section V.
III. LITERATURE SURVEY

A. Dependencies between distributed sites

Important dependencies/factors for task allocation are: Knowledge base, Technical resources and Communication and Coordination [5][17]. The research work [1] also identifies other dependencies such as scheduling strategy, state synchronization and synchronizing release schedule which affect the task allocation in varying degrees. Most of the dependencies except cultural and temporal also affect task allocation in a co-located environment, but their impact is more pronounced in distributed development. Distributed teams are not only separated by geographic distances but they also differ in knowledge base, technological expertise, organizational structures, temporal, communication and coordination aspects, socio-cultural norms and business processes [4][22]. All these dependencies/factors exercise considerable influence on the task allocation and their deliberation is essential.

B. Current task allocation strategies

Currently different task allocation strategies are being used in distributed environment. These are Modular Structures (Functionality based and Product based), Phase based structures (Process based), Functional Expertise based Structures, Customization based Structures and Follow the Sun Configuration (Overnight gain effect) [20][22][23][24]. It is evident that these strategies focus on only one criterion and ignore other important factors while assigning tasks to remote teams. Even if these strategies are used in conjunction with each other, some important factors like communication/coordination and cultural dependencies get ignored. Some surveys [20] also recommend that culture, product architecture, willingness to work and mutual trust must be included in our deliberations for task allocation. It is, therefore, important that a comprehensive strategy be worked out by including all relevant factors.

C. Current traceability support between views

Architecture is affected by communication and coordination needs of the organization. If tasks are allocated according to Conway’s law then the development view will change with change in communication and coordination needs. This change will also trigger change in other views such as deployment view, execution view and vice versa. There is a need to update linked views to incorporate the change effectively. This support can be given with help of traceability information. Availability of traceability links between various views will ensure the architecture’s interview alignment.

Work in the field of traceability between architectural views is carried out for different purposes such as concern evolution, requirement evolution and impact analysis etc. Traceability information between architectural views ensures consistency [16][25][26][27]. This linkage information can be used for task allocation/re-allocation in distributed teams [6] in a manner that it supports timely communication and coordination where necessary.

We have divided the literature survey on the basis of its focus of traceability support between UML diagrams and architectural views.

The focus of research [28][29] is requirement traceability. Research work [28] proposes an approach to provide traceability between requirements and UML diagrams using the Z Notation and XML. The UML diagrams included are use cases, class and sequence diagram. Traceability rules are defined to specify the above mentioned diagrams in Z language. The formal specification of the diagrams is then converted to XML schemas and traceability information is generated along with identification of missing requirements, inconsistent implementation and incomplete coverage. Traceability between use case, sequence and class diagram is also supported [29][30] via explicit saving of traceability links and via guided software production process respectively. A framework for the purpose of requirement tracing is presented [29]. The explicit link saving is performed via stereotypes and can help in change tracking as well as influence analysis. A supporting tool “Tracer” for implementing the framework is also presented. The guided production process moves from a requirement model (made using TRADE) to a conceptual model (made using Object Oriented method). The three views of the Object Oriented method include object model, dynamic model and functional model. These views include class diagram, state diagram, collaboration diagram and sequence diagram. We identify the responsibility in each use case as client (which invokes the responsibility) and implement the responsibility as server (which carries out responsibility). Responsibility is given via sequence diagram as it shows the participating classes in realizing the corresponding use case through interaction. The work of Lee et al. [31] provides traceability between sequence diagram, class diagram activity and collaboration diagram. The traceability support for activity diagram was not present in any of the previously mentioned work. The focus of the research is to evaluate the architecture for logical, behavior and performance issues. The approach works by converting UML artifacts to colored petri nets. More detailed traceability links are provided between use case model (activity and use case diagram) and object model (class and sequence diagram) with help of explicit link saving [32].
Traceability is provided for evolution of lower level models (sequence and class diagram) with respect to changes in higher level models (use case and activity models). Later evolution is supported by transversal of these links.

Traceability between use case and sequence diagram is also supported [33] via trace model and process description. The model also supports traceability with state diagram for the purpose of impact analysis of functional system requirements for embedded systems. It provides semi automatic traceability with help of prototype tool.

Traceability between class, component and deployment diagram is only performed by Bedir et al. [26]. The purpose of the work is to support concern traceability. A Concern Traceability Meta Model (CTM) is proposed which is used to model concerns, architectural elements (entity or relationship), and trace links between architectural views. The work is validated via case study of climate control system with the help of different change scenarios. The CTM is implemented using XML document type definitions (DTD). The instantiation are provided to support traces using XQuery. The traces are automatically identified using generic and specific queries written in XQuery and the results are shown in XML.

Our literature survey has revealed that there is very little traceability support between views with respect to the levels of traceability highlighted in [16]. The traceability links which are maintained or are implicitly present are between use case, class and sequence diagrams which are part of logical, process and use case view. Although the traceability support between above mentioned diagrams is present but the focus of research work is very narrow. Table 1 presents the results of the literature survey focusing on general and implementation information of each traceability technique/method.

IV. DISCUSSION

A survey of the current literature reveals that there is a linkage gap between different architectural views. Most of the work has been performed between use case, logical and process view. The work of Bedir et al. [26] whose focus is logical, implementation and physical view provides concern evolution but whether it is extendable to support full traceability across all views, for the purpose of our research is still to be seen. The physical view presented [26] concentrates on allocation of software units to hardware nodes disregarding work assignment style.

The literature survey also highlights the need for validation of traceability work to identify its usefulness for task allocation in distributed environment. Even if the traceability links within these views are identified it is not possible to relate them to work assignment view as this view is not designed in any of the viewpoint models. Moreover, the focus of research work surveyed ranges from concern evolution to impact analysis and architectural evaluation etc. thus providing traceability support only to solve the specific issues.
<table>
<thead>
<tr>
<th>Focus of Paper</th>
<th>Paper ID</th>
<th>Author(s) Year</th>
<th>Architectural View</th>
<th>UML Models Covered</th>
<th>Implemented via</th>
<th>validation</th>
<th>Tool Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[29]</td>
<td>T. Tsumaki, Y. Morisawa 2000</td>
<td>use case, process and logical view</td>
<td>Use case, Sequence and Class diagram</td>
<td>(Framework) Business Object Modeling and Design Methodology</td>
<td>Case Study</td>
<td>Proposed Tool (Tracer)</td>
</tr>
<tr>
<td>Concern Traceability</td>
<td>[26]</td>
<td>B. Tekinerdoga et al. 2007</td>
<td>logical view, implementation and physical view</td>
<td>Class, Component and Deployment diagram</td>
<td>(Meta Model) X-Query</td>
<td>Case Study</td>
<td>Research Tool M-Trace</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supporting Evolution in OO development</td>
<td>[32]</td>
<td>H. Omote et al. 2004</td>
<td>Use case, logical and process view.</td>
<td>Use case, Activity, Class and Sequence diagram.</td>
<td>Via Stereotypes of UML</td>
<td>Example</td>
<td>No</td>
</tr>
<tr>
<td>Impact analysis</td>
<td>[33]</td>
<td>A. Von 2002</td>
<td>Use case, logical and process view.</td>
<td>Use case, Class, Sequence and State diagram</td>
<td>(Trace Change Approach, Trace Model)</td>
<td>Experiment</td>
<td>Case Tool, St P/UML and Prototype Tool</td>
</tr>
</tbody>
</table>
V. CONCLUSION AND FUTURE WORK

Our literature survey reveals different task allocation strategies being used for assigning work to distributed sites. It also highlights various dependencies between distributed sites and traceability support between different views of viewpoint models. We find that distributed sites depend on each other for various things such as knowledge, process, module etc. but the current task allocation strategies do not take into account all these factors. Task allocation also depends on the communication and coordination needs of the teams which is depicted in architecture of the product.

Synchronizing different architectural views will result in informed and effective task allocation in a distributed environment. This synchronization is proposed by studying traceability linkage between view. The initial survey highlights inadequacy of linkage between architectural views. There is a need to identify/model the traceability links which would be specifically required for task allocation. We also need to model the work assignment view along with other views and provide synchronization between all of them. How this can be accomplished for an effective task allocation strategy will be seen in future.

REFERENCES