2012 IEEE Seventh International Conference on Global Software Engineering

A Tool Framework for Deriving the Application Architecture for Global Software Development Projects

Buğra M. Yildiz
Department of Computer Engineering
Bilkent University
Ankara, Turkey
e-mail: bugra@cs.bilkent.edu.tr

Bedir Tekinerdogan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
e-mail: bedir@cs.bilkent.edu.tr

Semih Cetin
Cybersoft Information Technologies,
METU Technopolis,
Ankara, Turkey
e-mail: semih.cetin@cs.com.tr

***Abstract* – In order to meet the communication, coordination and control requirements of distributed Global Software Development (GSD) teams, it is necessary to define a proper software architecture. Designing a GSD architecture, however, involves a multitude of design decisions that are related in different ways. As such, it is not trivial for the architect to design a system that meets the different GSD concerns. To assist the architect in designing a suitable GSD architecture we propose the tool framework *Global Architect*. The tool framework is based on a common meta-model for GSD and a question framework, which includes a predefined set of questions that are related to abstract design rules for designing GSD systems. Based on the answers provided to the questions of the question framework, the tool automatically selects and instantiates the necessary rules and generates the GSD architecture. *Global Architect* has been applied to design the GSD architecture for a real industrial project of Cybersoft, a leading GSD company in Turkey.**

Keywords-component; Architecture Modeling, Global Software Development, Model-Driven Development, Question framework

I. INTRODUCTION

Software architecture for a program or computing system consists of the structure or structures of that system, which comprise elements, the externally visible properties of those elements, and the relationships among them [12]. Software architecture forms one of the key artifacts in the entire software development life cycle since it embodies the earliest design decisions and includes the gross-level components that directly impact the subsequent analysis, design and implementation [14].

It is generally accepted that software architecture design plays a fundamental role in coping with the inherent difficulties of the development of large-scale and complex software. Research on architecture design in the last two decades has resulted in different useful techniques and approaches. Yet, in the software architecture design community the endeavor of software architecting seems to have been mainly focused on architecting in single systems. However, current trends in software engineering show that large software projects have to operate with teams that are working in different locations. The reason behind this globalization of software development stems from clear business goals such as reducing cost of development, solving local IT skills shortage, and supporting outsourcing and offshoring. There is ample reason that these factors will be even stronger in the future, and as such we will face a further globalization of software development. Global Software Development (GSD) is a relatively new concept in software engineering that can be considered as the coordinated activity of software development that is not localized and central but geographically distributed. Designing a proper architecture for GSD is important to meet the requirements for the communication, coordination and control of distributed GSD teams. Unfortunately, the global software engineering community seems to have less focused on these problems from a technical software architecting perspective.

In this paper, we present *Global Architect*, a tool framework for assisting the GSD architect in designing a proper architecture that meets the concerns of a global setting. Herewith, we define a GSD architecture as the gross-level structure of a software project that develops software across distributed sites. The GSD architecture, as such, embodies all the architectural elements and the relationships among them that together form the GSD project structure. We provide a meta-model that represents the key concepts of GSD. The tool framework is based on this meta-model for GSD and a corresponding question framework that aims to identify and derive the concerns of a GSD project. The question framework includes a set of predefined questions that can be used to provide an instantiation of the meta-model. Each instance of the meta-model is called an *application architecture* (GSD architecture). Both the meta-model and the question framework have been derived after a thorough analysis of the related GSD literature. Based on the answers provided to the questions of the question framework *Global Architect* automatically derives the related instance of the meta-model and, as such, provides the application architecture. The tool framework can be used to analyze different alternatives of application architecture for GSD projects.

The remainder of the paper is structured as follows. Section II describes the key concerns for architecting GSD. Section III describes the approach for deriving GSD application architecture. Section IV describes the meta-

978-0-7695-4787-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICGSE.2012.12

model for GSD and section VI the question framework. Section VI explains the tool framework, *Global Architect*, used for generating application architectures. Section VII gives a real case on which the approach is applied. Section VIII describes the related work and finally section IX concludes the paper.

II. ARCHITECTING GSD

Figure 1 shows the conceptual architecture for Global Software Development systems. GSD architecture usually consists of several nodes, or sites, on which different teams are working to develop a part of the system. The teams could include development teams, testing team, management team etc. Usually each site will also be responsible for following a particular process. In addition, each site might have its own local data storage.

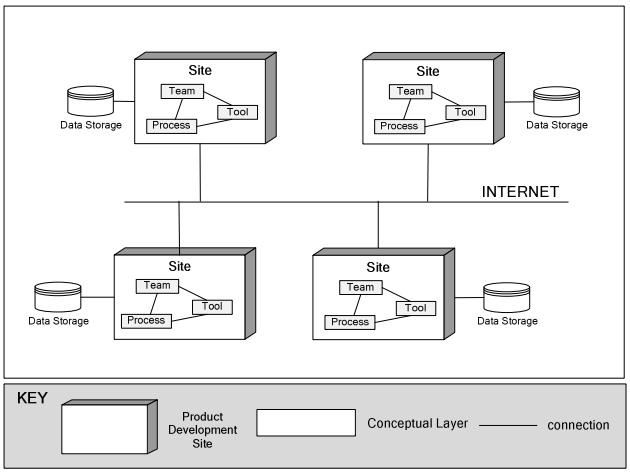

Figure 1. Conceptual Architecture for GSD

Different authors in the GSE community have identified *communication*, *coordination*, *control* and *development* as important concerns in GSD settings [1][4][10]. We explain these concerns in more detail below:

Development - the software development activities typically using a software development process. This includes activities such as requirements analysis, design, implementation and testing. Each project development site will typically address a subset of these activities.

Communication – communication mechanisms within and across sites. Typically different sites need to adopt a common communication protocol.

Coordination – coordination of the activities within and across sites to develop software according to the requirements. Coordination will be necessary to align the workflows and schedules of different sites. An important goal could be to optimize the development using appropriate coordination mechanisms.

Control – systematic control mechanisms for analyzing, monitoring and guiding the development activities. This does not only include controlling whether the functional requirements are performed but also which and to what extent quality requirements are addressed.

In fact, each of these concerns requires further in-depth investigation and has also been broadly discussed in the GSD community. Within the context of this paper we are interested in the impact of these concerns on the architecture of the GSD. Designing a proper architecture for GSD systems is important to meet the requirements for the communication, coordination and control of distributed GSD teams. However, designing the GSD architecture involves many different concerns and usually is not a trivial process. The GSD architecture will need to be different for different parameters such as team size, data storage, adopted processes, migration of the teams, and communication protocols, etc. Unfortunately, no systematic and dedicated approach exists in assisting the GSD architect to derive the GSD architecture.

III. APPROACH FOR DERIVING THE APPLICATION ARCHITECTURE

GSD architecture can be described as a model that is instantiated from a common GSD meta-model. For deriving an architecture the architect will be supported by a question framework. Figure 2 shows the more detailed relations among the concepts that explain this process.

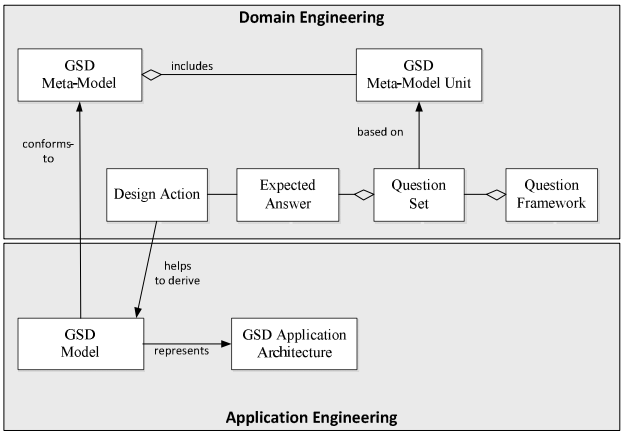

Figure 2. Conceptual Model defining the relation between question framework meta-model and application architecture

In this context, an application architecture can be considered as a representation of an instance of a meta-model. In practice, different multiple instances can be derived from a meta-model based on the project requirements. To capture these project requirements, we adopt the use of Question Framework which includes a set of Question Sets. Question Sets include the expected answers for which related design actions are defined. The design actions can be used to derive an instance of the meta-model and later on map this to the application architecture.

Based on the model in Figure 2, the process consists of two basic steps, the *domain engineering* phase and the *application engineering* phase. In the domain engineering phase, based on a thorough literature study, first the meta-model for GSD architecture is defined. The meta-model defines a reference architecture for application architecture

that can be derived from it. The meta-model itself consists of several units. In addition to the meta-model, the question framework is defined to support the derivation of the application architecture. The question framework includes a number of question sets that include questions. Typically a question set is defined for each unit of the meta-model. For each question in the question set the expected set of answers is defined. Finally, the set of reference design actions are defined.

In the application engineering phase the set of answers based on expected answers are derived. Using the application answers and the reference design actions the set of application design actions are defined. These design actions are used to instantiate the GSD meta-model and subsequently derive the GSD application architecture. The complete process is given in Figure 3.

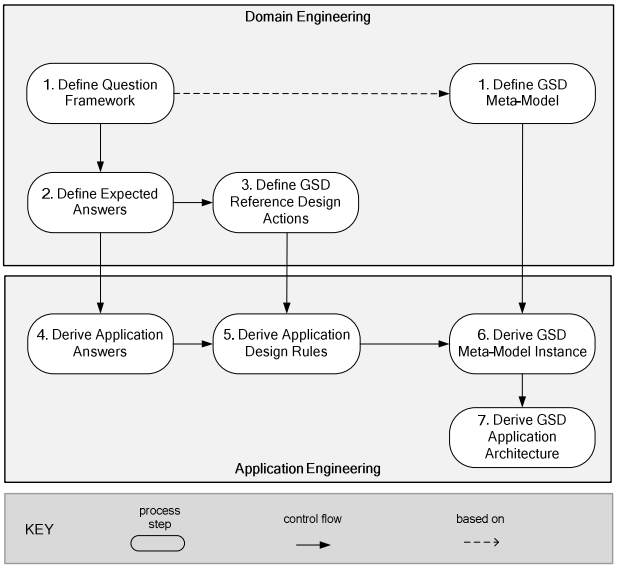

Figure 3. Process for deriving application architecture

IV. META-MODEL FOR GLOBAL SOFTWARE DEVELOPMENT

Based on the literature of GSD, we have defined a meta-model for GSD that defines the concepts and their relations to derive application architecture. Since the meta-model is quite large we have decomposed it into six meta-model units. Each of these meta-model units includes semantically close entities and addresses the four concerns that have been defined in section 2. The meta-model units are also overlapping; the meta-model elements with the same name in different units refer to the same meta-model elements. The meta-model units are *Deployment*, *Process*, *Data*, *Communication*, *Tool* and *Migration*. We explain these units below.

1. Deployment Unit

Deployment Unit concerns the deployment of the teams to different sites. The meta-model (abstract syntax) of this unit is shown in Figure 4.

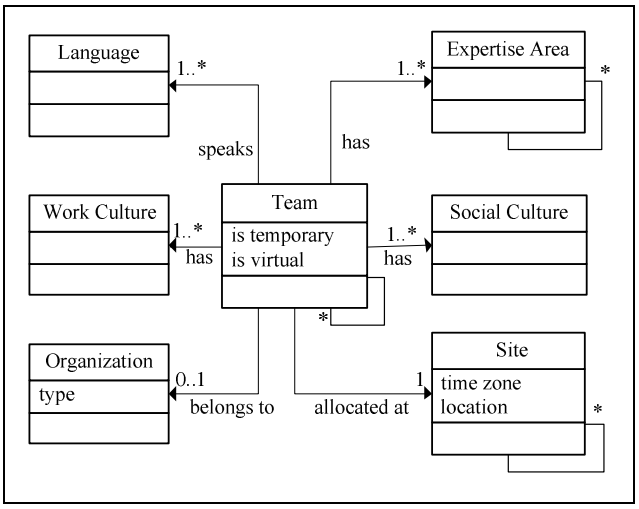

Figure 4. GSD Meta-model: Deployment Unit

Team is the primary essential entity in the complete meta-model and is defined as a group of persons that work together to achieve a particular goal. A *Team* may be organized in a temporary way that it will be dismissed after its function is complete. *Team* is allocated at a particular *Site*. *Site* may to a country, city or a building where a Team works at. Location attribute determines where *Site* is placed in the world. Time zone shows the local time of *Site*. *Teams* may belong to different types of *Organizations*, such as commercial organizations, subcontractors or non-profitable organizations such as open source communities. *Teams* can be from different countries and depending on the society they are in, they may have different *Social Cultures*. Like *Social Culture*, *Team*'s background including work experience, the time that members work together, their habits are captured by *Work Culture* entity.

Expertise Area, *Team* and *Site* can be further decomposed into sub-parts. For example, a Software Team may consist of sub-Teams each responsible for Design, Implementation, Testing and Integration.

2. Process Unit

Process Unit concerns the different kind of processes in GSD. The meta-model (abstract syntax) of this unit is shown in Figure 5.

Process is defined as a planned set of activities that aims to provide some service. *Teams* participate in *Process* in order to provide some service. Service is defined with *Function*. A *Function* can be any service during software development process that requires some *Expertise Areas* such as software development, architecture design, business management, requirements elicitation and so on. *Coordination* is also a *Function* that should be provided for coordinating several *Teams*' activities. A *Process* consumes or uses several different *Data Entities* and also creates other *Data Entities* for providing targeted *Functions*. For supporting activities defined in *Process*, *Teams* need to

communicate with each other. These *Communications* support *Process*. *Process* concept is further specialized into Workflow, Business Process and Development Process (not shown in figure).

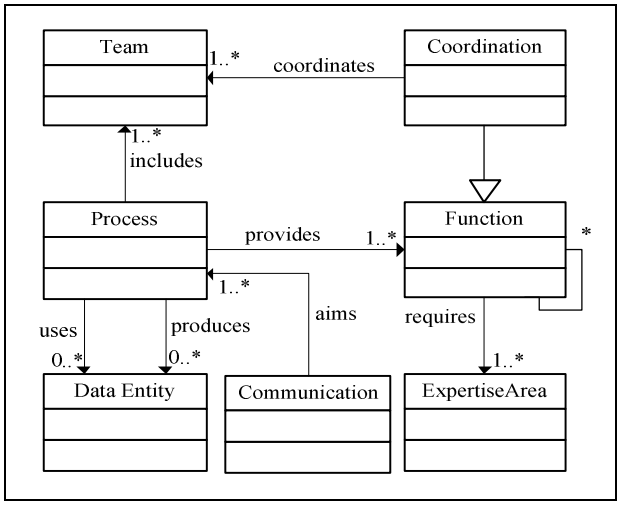

Figure 5. GSD Meta-model: Process Unit

3. Data Unit

Data Unit is for representing ownership and physical deployment of software development data. The meta-model is shown in Figure 6.

Data Entity is the fundamental entity of this viewpoint. It represents any piece of data: digital, textual or informal piece of information such as notes taken by developers, telephone calls that are usually not recorded. *Data Entity* has size whose unit is defined by size type; for example, a 120-page report, 6 minutes of voice record, 2 gigabyte of digital data. *Creation date* and *last update date* show the history of *Data Entity*. *Data Entity* has *Actual Type* where *Actual Format* can be one of predefined formats (video, sound, text, picture and complex-*Data Entity*) or some designer defined format. If *Data Entity* is digital, then in addition to *Actual Format*, it has a *Digital Format*. *Data Entity* may be implemented in one or more *Languages.*

Data Entity is stored in *Data Storage*. *Data Storage* corresponds to any object in the real world that can store information. For example, some textual document is stored in paper form, or it is stored in a voice record, or it is stored digitally in the format of some text editor. *Data Storage* has ability to store some *Actual Types* and if it can store digital data then it can support some *Digital Types* also. A *Data Storage* instance is owned by one or more *Teams* and it can be located in one *Site* or may be distributed over several *Sites* like distributed databases.

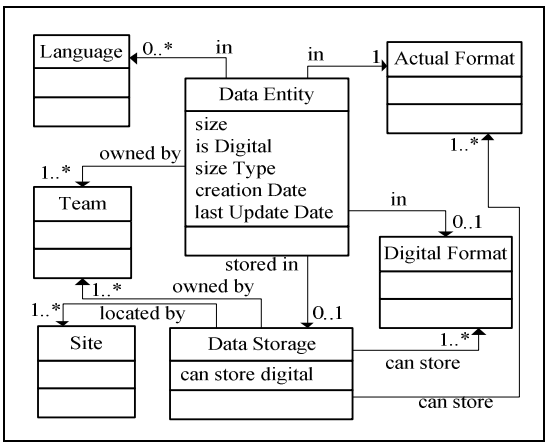

Figure 6. GSD Meta-model: Data Unit

4. Communication Unit

Communication Unit focuses on the representation of both formal and informal communication activities between *Teams*. The meta-model is shown in Figure 7.

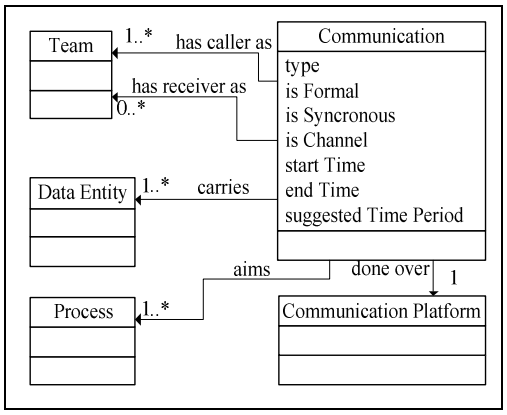

Figure 7. GSD Meta-model: Communication Unit

Communication is done over *Communication Platform* in the context of *Process* and it can be an instance of sudden/event-based communication activity like a telephone call or a continuous communication channel such as a discussion forum. *Type* attribute is for representing in which way *Communication* takes place such as email, phone call, face-to-face chat and so on. *Suggested time period* is an important attribute for GSD since Teams work in different time zones, some *Communication* channels can be used effectively in a defined time period. For example, phone calls should be done during the hours when both sides are in or around their work hours.

Communication has two sides which are caller and receiver. Generally speaking, caller starts communication and receiver is the one who is dialed by the caller. For example, an email sender is classified as caller and receiver is the one who receives email. Sometimes, there can be multiple callers such as video conferences or there can be multiple receivers such as discussion forums. It is also

possible that caller and receiver are the same such as a planned meeting. For all cases, caller and receivers are considered as *Teams* in this viewpoint. While *Teams* communicate, one or more *Data Entities* are carried in the context of *Communication*.

5. Tool Unit

Tool Unit captures the details of tools used by *Teams* for communication and providing *Functions*. The meta-model is shown in Figure 8.

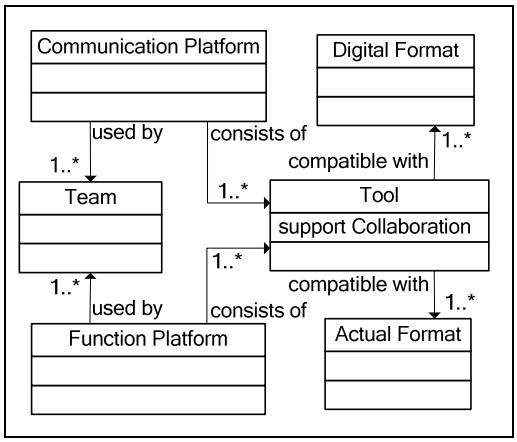

Figure 8. GSD Meta-model: Tool Unit

Tool is compatible with one or more *Actual Format* and *Digital Format*. Platform is the set of *Tools* used by *Teams* for communication or providing some functions. Depending on the purpose, the platform is defined as *Function Platform* or *Communication Platform*.

6. Migration Unit

Migration Unit concerns the migration and traveling of the teams during GSD activities. These travels are especially needed in the first and final phases of the projects to ease and support coordination and integration. The meta-model is shown in Figure 9.

Migration is executed by one or more *Teams* from *Site* to *Site* at a particular date. In a *Migration, Teams* may carry *Data Storage* such as documents, digital data containers and so on. *Migration* is executed in the context of *Process*.

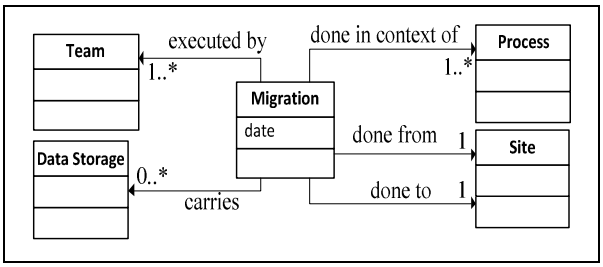

Figure 9. GSD Meta-model: Migration Unit

V. Defining The Question Framework for GSD

In order to support the derivation of application architecture, we have defined a question framework dedicated for GSD. The question framework is divided into six question sets. Each set corresponds to one of the six meta-model units as described in the previous section. Like meta-model units, question sets also have intersecting concepts. The questions are numbered in a particular order to guide the proper derivation of the application architecture. As an example the question set for the deployment meta-model unit is shown in Table 1.

A question set consists of a number of questions. For example for the Deployment Unit we have identified 19 questions related to the elements in the corresponding meta-model unit. Each question is defined as a tuple consisting of the *index number of the question*, the *question* itself, the *expected answer* and the *related design action*.

The expected answers are specific to the questions and might include values of primitive data types (such as integer) or require more complicated descriptions. Each expected answer can trigger one or more *design actions.* A design action defines a CRUD (create, read, update and delete) action to create, read, update or delete the elements of a model representing the application architecture. The actions on which these CRUD operations can apply are in principle *Model, Model Attribute, Model Operation,* and *Association.* Based on this we have defined all the possible combinations for CRUD operations. For example, we have defined the following types of design actions:

- *Create Model Instance*
- *Read Model Instance*
- *Delete Model Instance*
- *Update Model Instance*
- *Create Association*
- *Read Association*
- *Delete Association*
- *Update Association* -
- Etc.

The design action is defined as a tuple consisting of <operation><model element>. The <operation> part represents one of the four CRUD operations. The part <model elements> represents the model or model elements. For example, the design action related to the first question of Table 1 is defined as <Create Model><Site> which implies the creation of the Model Site. Similarly, the description <Create Association><Site Site> of question 3 denotes the creation of an association between Site model elements.

Table 1. Question Set for Deployment Meta-Model Unit

No	Question	Expected Answer	Design Action
1	How many Sites are there in this GSD project?	Positive integer.	<Create Model> <Site>
2	Enter Site details.	Name, description and location are given in free text format. Time zone is selected from time zones in the world.	<Update Model-Attribute> <Site>.
3	Select parent Site for each Site.	A defined Site or null value is selected for each Site.	<Create Association> <Site Site>
4	Which Languages are used?	Selections are done from languages spoken in the world.	<Create Model> <Language>
5	What are Organization types?	Free text. Ex: Comercial, Open Source Development, etc.	<Guide> Determine the list of types for Organizations.
6	How many Organizations are there in this GSD project?	Positive integer.	<Create Model> <Organization>
7	Enter Organization details.	Name and description are given in free text format. Type is chosen from the values given in Question 6's answer.	<Update Model-Attribute> <Organization>
8	Define Social Cultures.	Name and description for each instance that is wanted to be created are given in free text format.	<Create Model> <Social Culture>
9	Define Work Cultures.	Name and description for each instance that is wanted to created are given in free text format.	<Create Model> <Work-Culture>
10	How many Expertise Areas are there in this GSD project?	Positive integer.	<Create Model> <Expertise Area>
11	Enter Expertise Area details.	Name and description are given in free text format.	<Update Model-Attribute> < Expertise Area>
12	Select parent Expertise Area for each Expertise Area.	A defined Expertise Area or null value is selected as parent for each Expertise Area.	<Create Association> <Expertise Area Expertise Area>
13	How many Teams are there in this GSD project?	Positive integer.	<Create Model> <Team>
14	Enter Team details.	Name and description are given in free text format. isTemporary and isVirtual attributes are given in boolean values.	<Update Model-Attribute> <Team>
15	Select Site and Organization for each Team.	A defined Organization and Site or null value is selected for each Team instance.	<Create Association> <Team-Site Team-Organization>
16	Select Work and Social Cultures for each Team.	At least one defined Work Culture and Social Culture are selected for each Team.	<Create Association> < Team-Work Culture>; <Create Association> < Team-Social Culture>
17	Select Languages that each Team able to speak.	At least one defined Language defined before is selected for each Team.	<Create Assocication> <Team-Language>
18	Select Expertise Areas for each Team.	At least one defined Expertise Area is selected for each Team.	<Create Association> <Team-Expertise>
19	Select parent Team for each Team.	A defined Team or null value is selected as parent for each Team.	<Create Association> <Team-Team>

Once the questions are answered and the related design actions are collected we can derive the application architecture. In principle, it is possible to create an application architecture by using the meta-model and the question framework manually. However, answering all the questions and keeping track of the required design actions can be a cumbersome and error-prone activity. In addition, since the relation among question, expected answers and design actions are precisely defined, we have provided automated support for the approach in the tool framework, *Global Architect*. We explain this tool in the next section.

VI. THE TOOL FRAMEWORK: GLOBAL ARCHITECT

We have developed the tool framework *Global Architect* that implements the meta-model and the question framework described in the previous sections. The tool is implemented in the Java environment. A snapshot of the tool is shown in Figure 10. The *menu bar* at the top provides basic functionality such as creating, importing/loading, exporting or editing architectural models. The architect creates a new model or imports an existing model before answering the related questions of the question framework. The tool includes 6 question sets of the question framework. These question sets are shown as tabs in the snapshot. The questions related to each question set are shown on the left panel. The architect can navigate through each of these questions. The tool itself defines the required ordering to answer questions for satisfying the question dependencies. The selected question with the answer fields are shown in the middle panel.

The answers to the questions result in an update of the current model, which is an instance of the meta-model that we have defined in the previous sections. The view on the partial model is shown on the right panel. In the snapshot, the question relates to the structure of teams in GSD, and likewise the *Team hierarchy* part of the model is shown in the right panel.

Each answer to a question triggers a design action that updates the underlying model of the architecture. *Global Architect* keeps track of the answers given to the questions and manages the dependencies among questions. An answer to a question can impact multiple views because the same model element might need to be changed. The tool keeps track of these changes on overlapping entities of different views and, as such, the consistency among views is ensured. As an example, the teams are defined in the model by answering the question 14 and the tool limits the architect to use these defined teams in later questions. Similarly, the architect is forced to use these teams while answering other question sets since *Team* is a common entity in different views.

After answering the questions the architect can export the model of the application architecture in a XML-like textual form that we specifically defined. An (partial) example textual representation of the application architecture is given in Figure 11.

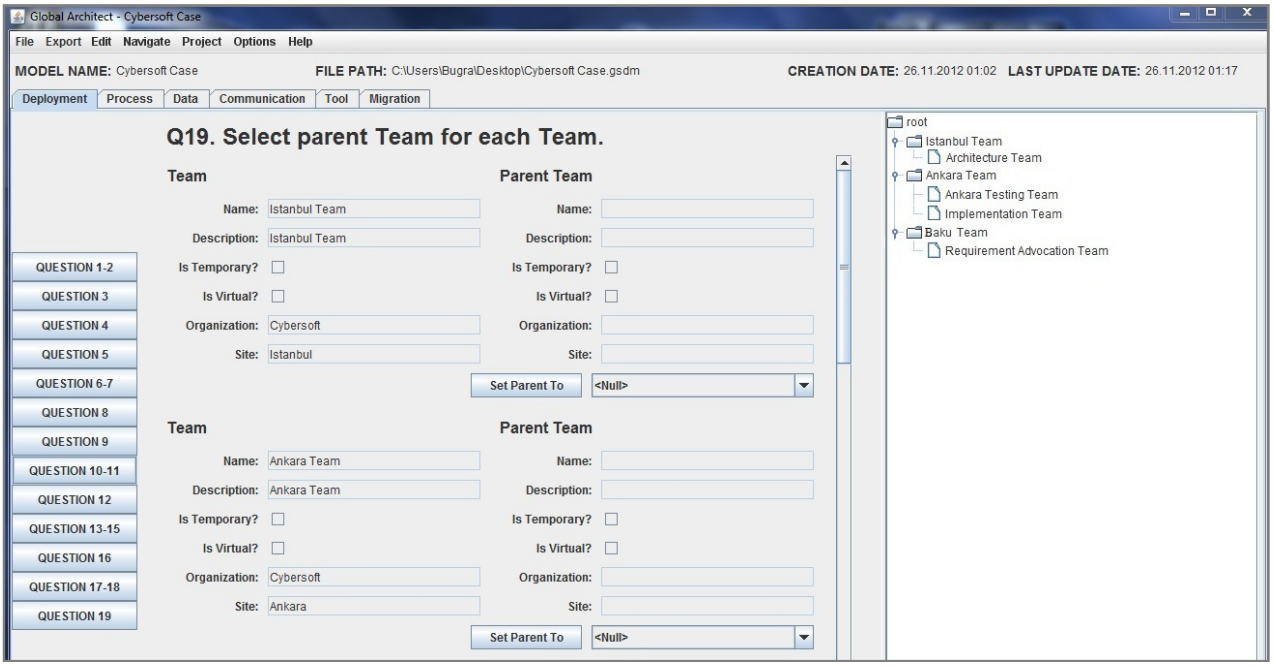

Figure 10. Snapshot of Global Architect Tool

For the generation of visual models of the application architecture from these textual representations, we have adopted model-driven development techniques. The model-transformation pattern we applied for this procedure is shown in Figure 12.

```
<Site id=8 name="Ankara" description="Ankara" timezone="Turkey" location="Ankara"
    childrenList=[
    ]
    teamList=[
        <Team id=22 name="Ankara Team" description="Ankara Team" isVirtual= isTemporary=
            languageList=[
                <Language id=9 name=English/>
                <Language id=11 name=Turkish/>
            ]
            organization=<Organization id=12 name="Cybersoft" description="Cybersoft" type="Commercial" />
            socialCultureList=[
                <SocialCulture id=13 name="Turkish Culture" description="Turkish Culture"/>
            ]
            workCultureList=[
                <WorkCulture id=15 name="Cybersoft Ankara" description="Cybersoft Ankara"/>
            ]
            expertiseAreaList=[
                <ExpertiseArea id=17 name="Architecture Design" description="Architecture Design"/>
                <ExpertiseArea id=19 name="Local Testing" description="Local Testing"/>
                <ExpertiseArea id=20 name="Implementation" description="Implementation"/>
            ]
        />
        <Team id=26 name="Ankara Testing Team" description="Ankara Testing Team" isVirtual= isTemporary=
            languageList=[
                <Language id=9 name=English/>
                <Language id=11 name=Turkish/>
            ]
            organization=<Organization id=12 name="Cybersoft" description="Cybersoft" type="Commercial" />
            socialCultureList=[
                <SocialCulture id=13 name="Turkish Culture" description="Turkish Culture"/>
            ]
            workCultureList=[
                <WorkCulture id=15 name="Cybersoft Ankara" description="Cybersoft Ankara"/>
            ]
            expertiseAreaList=[
                <ExpertiseArea id=19 name="Local Testing" description="Local Testing"/>
```

Figure 11. Example Textual Application Architecture derived using GSD *Global Architect* Tool

The source of the transformation consists of the textual model file that has been derived after answering the questions and triggering of the design actions in the *Global Architect* tool. For the target language we have developed a UML deployment meta-model. The transformation definition in Figure 12 defines the mapping of the GSD model elements to the elements of the deployment model. The mapping is developed as an Eclipse plug-in [8] in which we have used the Eclipse Graphical Modeling Framework (GMF) [7] for developing the graphical representation. After the automatic generation of deployment diagram, the user can modify the diagram manually.

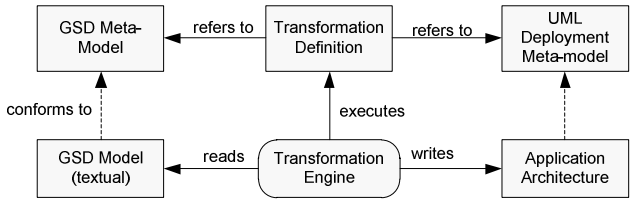

Figure 12. Model transformation pattern for mapping textual model to UML deployment view

VII. CASE STUDY

The question framework and the concepts behind this work have been tested on a real case at Cybersoft. The company is the leading software house in Turkey having dedicated branches in Ankara and Istanbul. Both of these branches are conducting enterprise scale software projects on their own. However, during past two years the company is urged to have distributed teams that have to contribute to the other branch's work. The distribution of work between

different branches required re-modeling of the distributed software development architecture.

The GSD project chosen concerns the development of a banking and insurance system that is distributed over two countries Turkey and Azerbaijan. *Global Architect* has been used to derive the GSD architecture for this distributed software development project at Cybersoft. For this the project managers and the architects of the project have answered the questions of the question framework in *Global Architect.* For example, the answers to questions for *Deployment Unit* revealed the need of creation of vertical teams for different expertise areas. The company's headquarters is located in Ankara where the implementation and testing teams work. The architecture design process is executed by another team of the company located in Istanbul branch. Specific to this project, the company allocated a requirement advocation team that is working in Baku, Azerbaijan. The tool framework is used to answer the questions of the deployment question set and the textual representation of the architectural model is produced by the tool given in Figure 11. The textual model has been provided as an input to the model generator as shown in Figure 12. The derived generated visual architecture is presented in Figure 13.

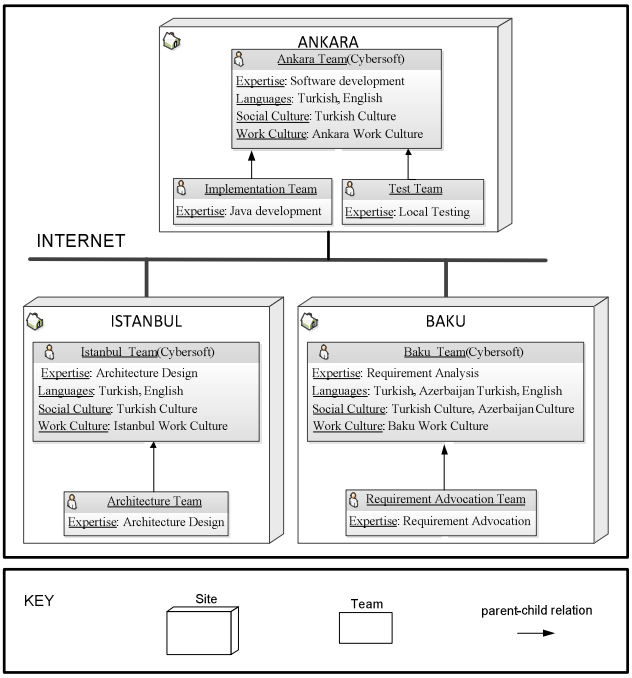

Figure 13. Generated Deployment Architecture for the Cybersoft case

Answering the questions for *Process Unit* created new process definitions such as synchronization of the test cases between sites, multi-site release management, and distributed work item management, all of which do not exist during single site development.

As *Data Unit* related questions are answered, the tool introduced wiki-based distributed requirement store, teleconference voice records, and digitized paper document repository all of which are newly brought by the distributed way of working.

The answers to *Communication Unit* questions yielded several new communication channels such as voice links, wiki, forums and knowledge-based cheat sheets. Moreover, these answers also introduced new communication-based processes such as mapping requirements to voice records or version management based on the changes on wiki items. They do not exist in the conventional way of working at Cybersoft.

The questions for *Tool Unit* have been answered and this leads to the introduction of new tools such as wiki, Web-based teleconference managers, Web-based instant screen snapshot replicatiors, all of which do not exist before the distributed way of working.

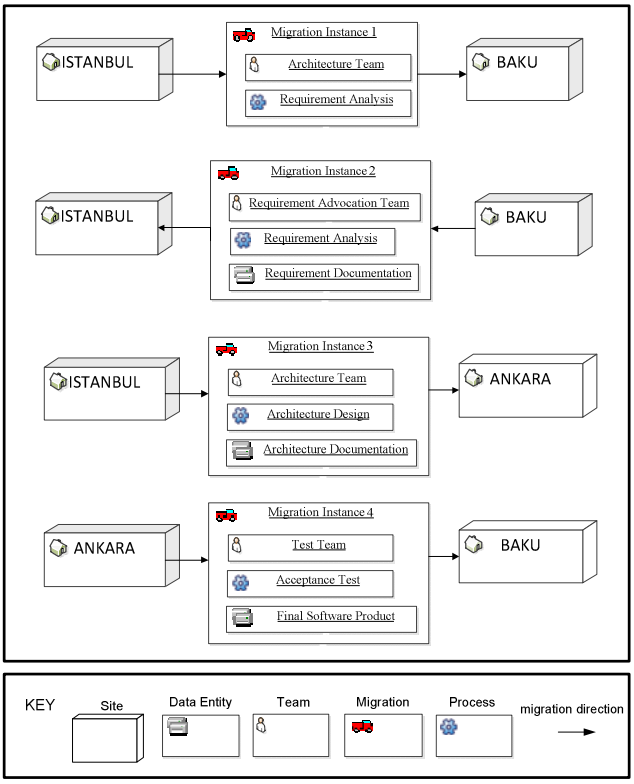

Figure 14. Generated Migration Architecture for the Cybersoft case

The answers to *Migration Unit* questions created brand new tasks to manage the mobilization of teams from time to time. During these face-to-face communication sessions, new data units such as wiki synchronization reviews, work redistribution logs, and distributed team efficiency statistics are all introduced which do not exist before the distributed way of working. The face-to-face communication sessions required to travel of the teams working in different branches to the other cities for completing some processes. The architect and the requirement advocation teams travel between Istanbul and Baku to clarify the requirements of the system to be developed. It is also needed for the architect

team to visit Ankara branch in order to transfer the system architecture and architectural decisions to the lead developers working in the headquarters. In the final phase of the project, the test team needs to go from Ankara to Baku for executing the acceptance tests. These travels are shown in Figure 14. The figure represents the visual model of the migration architecture derived using the tool framework and the text-to-visual model transformation pattern.

The project managers and the architects have derived a variety of application architectures with different perspectives by answering the question sets considering several concerns. By comparing these alternatives, they realized different needs. The results from the case study showed that *Global Architect* is useful in terms of the representation and derivation of the application architecture from different perspectives. In addition the tool helped to reveal the needs in different GSD concerns for the project and derive different architecture alternatives.

VIII. Related Work

Notably, architecting in GSD has not been widely addressed. The key research focus in the GSE community seems to have been in particular related to tackling the problems related to communication, coordination and control concerns. Clerk et al. [6] report on the use of so-called *architectural rules* to tackle the GSD concerns. *Architectural rules* are defined as "principles and statements about the software architecture that must be complied with throughout the organization". They have defined four challenges in GSD: time difference and geographical distance, culture, team communication and collaboration, and work distribution. For each of these challenges they list possible solutions and describe to what extent these solutions can be expressed as architectural rules. The work of Clerk et al. aims to shed light on *what* kind of architectural rules are necessary to guide the GSD. We consider our work complementary to this work. In our work the design actions that relate to the expected answers of questions are defined as design actions.

In our work the notion of question framework plays an important role to develop the architecture. The notion of question framework has been addressed before by several authors. To the best of our knowledge question frameworks have only been used as a means for evaluating architectural descriptions, and not for supporting the development of the architecture. For example, in [9] Hämäläinen and Markkula have introduced a question framework for evaluating quality of architectural descriptions. The quality evaluation question framework is organized based on *stakeholder and purpose orientation*, *content quality*, *presentation and visualization quality*, and *management of documents*. The *stakeholder and purpose orientation*, is defined for evaluating how well documents are focused on their purpose and on the stakeholders using them. The *content quality* is used to evaluate the quality of the information included in the documents. The *presentation and visualization quality*, is used for evaluating how well information is presented in the documents. Finally, the *management of documents* is used for evaluating architecture description quality, from the point of view of processes and practices. Similarly, Nord et al. [15] propose a structured approach for reviewing architecture documentation (AD) which also builds on question frameworks. The approach is centered on the stakeholders of the artifact, and aims to engage and guide them to assure that the architecture documentation meets their quality concerns.

A common practice is to model and document different architectural views for describing the architecture according to the stakeholders' concerns [5][11][12][13]. An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern. Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. An architectural framework organizes and structures the proposed architectural viewpoints. Different architectural frameworks have been proposed in the literature. Examples of architectural frameworks include the Kruchten's 4+1 view model [13], the Siemens Four View Model [11], and the Views and Beyond approach (V&B) [5]. In our work we derive an application architecture that represents the deployment view of the system. This view appeared to be one of the most useful views since it is able to depict the multi-site character of GSD. However, we could easily consider other views such as decomposition view or uses view. In that case we need to provide new implementations for mapping between the meta-model instance and the target view.

Tool support has been named as one of the important challenges for GSD since it requires making software development tools and environments more collaborative [16]. Booch and Brown have introduced the vision for *Collaborative Development Environment (CDE)*, which is defined as "a virtual space wherein all the stakeholders of the project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts" [1]. A number of efforts have been carried out to support the idea of CDEs. *Collab.net* [3] is a commercial provider of CDEs, offering facilities for configuration management, bug tracking, task management and discussions. Spanjers et al. [17] discuss the system *SoftFab,* which automates the build and test processes in the context of multi-site projects. Caroll et al. [4] define the tool *Jazz* which supports rich synchronous communication, and promotes mutual awareness of coding activities within a development team.

In the context of tool support, Whitehead [18] has presented a survey on existing collaboration support tools in software engineering. Whitehead distinguishes among four broad categories of tool support to support collaboration in software engineering: *Model-based* collaboration tools for representing the adopted models; *Process support* tools for representing software development process; *Awareness tools* for informing developers about the ongoing work of others and to avoid conflicts; *Collaboration infrastructure* to support data and control integration and likewise support interoperability.

Despite the clear need and benefits of the existing CDE tools, it appears that most of the work on CDE has focused on the collaboration concern and less on the development part. Further the tools that address development primarily focus on collaborative coding and relatively little attention has been paid to architecture design. There seems to be a general agreement that more research is needed in this domain. Our approach and tool can be considered as part of the efforts for enhancing CDE for design of GSDs.

IX. CONCLUSION

Different challenges have been identified to set up a Global Software Development environment. Our literature study on GSD showed that in particular the challenges of development, communication, coordination, and control of GSD are addressed in the GSD community but less focus has been provided on the modeling, documentation and analysis of architecture for GSD. In this paper we have focused on the architecture design of GSD. Designing architecture for single systems is hard. Designing architecture for GSD is even more difficult due to the additional concerns for communication, coordination and control of distributed GSD teams.

To support the architect in designing a proper GSD architecture we have provided a tool framework based on a question framework which has the purpose of identifying and deriving the concerns of a GSD project. The question framework includes important questions related to concerns for designing the architecture. Despite traditional usage of question frameworks as evaluation tools, we have defined and used the question framework to derive the application architecture for GSD. For this we have defined the possible design actions for the expected answers of the questions in the question framework. The answers to the questions triggers design actions which make updates on the architectural model. The model conforms to a meta-model that we defined for GSD based on a thorough literature study. The meta-model captures the key concerns of GSD architectures including *deployment*, *process*, *data*, *communication*, *tool* and *migration.* For each of these concerns we have defined a dedicated question set that is organized in the overall question framework. To support the question framework we have developed the tool *Global Architect*, builds on the defined meta-model and implements the question framework. The tool framework has been validated in a real industrial case study. Using *Global Architect* we were not only able to generate the architecture but also able to reason about different alternatives.

In our future work, we plan to extend the functionality of the tool framework by providing online help, supplementary tutorials and consolidated visualization of the final complete architecture. Also we aim to apply *Global Architect* for other GSD projects.

REFERENCES

[1] P. J. Ågerfalk , B. Fitzgerald, Introduction, Communications of the ACM, v.49 n.10, October 2006 .

[2] G. Booch and A. Brown. *Collaborative Development Environments*. Advances in Computers Vol. 59, Academic Press, August, 2003.

[3] Collab.net, http://www.collab.net, http://sourceforge.net.

[4] M. Carroll and S. Sprenkle. *Coven: Brewing Better Collaboration through Software Configuration Management*. ACM SIGSOFT Foundations of Software Engineering, pp. 88-97, 2000.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. *Documenting Software Architectures: Views and Beyond.* Second Edition. Addison-Wesley, 2010.

[6] V. Clerc, P. Lago, H. van Vliet. *Global Software Development: Are Architectural Rules the Answer?* In: Proc. of the 2nd International Conference on Global Software Engineering, pp. 225–234. IEEE Computer Society Press, Los Alamitos, 2007.

[7] Eclipse Graphical Modeling Framework, http://www.eclipse.org/gmf/, accessed February 2012.

[8] Eclipse official web site, http://www.eclipse.org, accessed 2011.

[9] N. Hämäläinen & J. Markkula. Quality Evaluation Question Framework for Assessing the Quality of Architecture Documentation. In: Proceedings of International BCS Conference on Software Quality Management. University of Tampere, SQM, 2007.

[10] J.D. Herbsleb and D. Moitra. Global Software Development. IEEE Software, March/April, p. 16- 20, 2001.

[11] C. Hofmeister, R. Nord, and D. Soni. Applied Software Architecture. Addison-Wesley, NJ, USA.

[12] [ISO/IEC 42010:2007] International Organization for Standardization & International Electrotechnical Commission. Systems and software engineering—Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010). (identical to ANSI/IEEE Std1471–2000), July 2007.

[13] P. Kruchten. The 4+1 View Model of Architecture. IEEE Software, 12(6):42–50, 1995.

[14] A.J. Lattanze. Architecting Software Intensive Systems: A Practitioner's Guide, Auerbach Publications, 2009.

[15] R. Nord, P. Clements, D. Emery, R. Hilliard. A Structured Approach for Reviewing Architecture Documentation, Technical Note, CMU/SEI-2009-TN-0302009, SEI-CMU, 2009.

[16] B. Sengupta, S. Chandra, V. Sinha. A research agenda for distributed software development, In Proceedings of the 28th international conference on Software engineering, pp. 731-740, 2006.

[17] H. Spanjers, Ter, B. Graaf, M. Lormans, D. Bendas, R. V. Solingen. Tool Support for Distributed Software Engineering, in Proc of: International Conference on Global Software Engineering, 2006. ICGSE '06., pp. 187-198, 2006.

[18] J. Whitehead, *Collaboration in Software Engineering: A Roadmap*, In FOSE '07: 2007 Future of Software Engineering, pp. 214-225, 2007.