DPTool: A Tool for Supporting the Problem Description and Projection

Xiaohong Chen[1], Bin Yin[1] and Zhi Jin[1,2]
[1]Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China
[2] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Peking University, Beijing, China
{chenxh, ybziwen, zhijin}@amss.ac.cn

Problem Frames (PF) approach is prospective for describing and analyzing software problems [1]. Problem decomposition is fundamental for managing problem size and complexity in RE and 'projection' has been argued to be an effective technique for analyzing and decomposing complex problems in PF. However, problem analysis in PF approach is still an empirical, tedious, and subjective process, and it heavily depends on the analysts' experiences. A suitable tool is highly demanded for providing guidance of the problem description and projection [2].

We had brought scenario into PF and gave a scenario based problem projection in [3]. A scenario is designated to express how interactions can satisfy the requirements. By combining the elemental concepts in PF as well as the scenario based approaches, we developed a scenario-extended problem ontology as the concept model of the problem description and projection. We proposed to use the well-formed scenarios as the projected aspect of the problem projection. A problem projection operator is defined and with this a scenario-extended problem description and projection process is figured out.

Based on [3], this paper gives a supporting tool (named DPTool) for describing the problem and performing the problem projection. DPTool is a graphical tool which provides various editing, checking and verifying functions for problem description elements, and supports performing the problem projection.

I. PROCESS OF PROBLEM DESCRIPTION AND PROJECTION

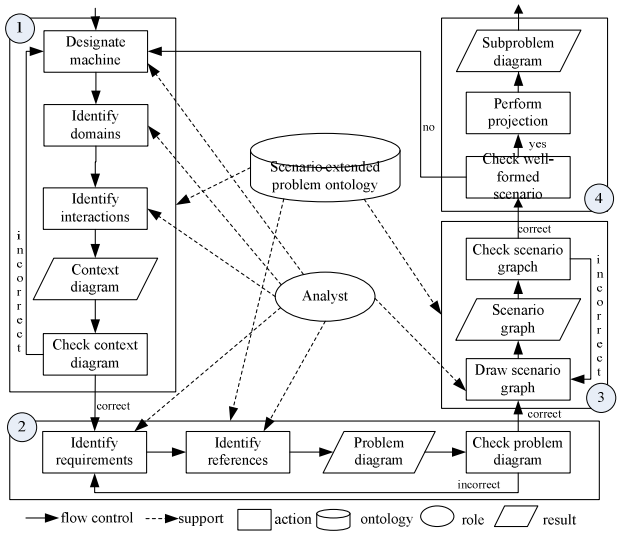

Figure 1. Process of problem description and projection

Fig. 1 gives the problem description and projection process. It includes 4 steps. They are drawing the context diagram, drawing the problem diagram, constructing the scenario graphs and performing the problem projection. The details of each step are described as follows.

Step 1: Drawing the context diagram. This step helps the analysts locate the boundary of the software problem. The result of this step is the context diagram of the to-be built machine. It has 4 sub-steps:

a) Giving a name to the to-be built machine;
b) Identifying the domains that will interact with the machine;
c) Identifying the interactions between the machine and these domains;
d) Checking the context diagram to see if it is accordance with the pre-defined constraints in the ontology about the machine, the domains and the interactions; if it is correct, go to the next step; otherwise, show the message about the errors and return to a) to ask for re-drawing.

Step 2: Drawing the problem diagram. This step is for helping the analysts to define the requirements. The result of this step is the problem diagram of the to-be built machine. It has 3 sub-steps:

a) Identifying the requirements that are desired to be satisfied by the problem;
b) Identifying the requirement references between the requirements and domains;
c) Checking the problem diagram to see if it is accordance with the predefined constraints in the ontology about the requirements; if it is correct, go to the next step; otherwise, show the message about the errors and return to a) to ask for re-drawing.

Step 3: Constructing the scenario graphs. This step is for helping the analysts to define the requirement realization. The result is a set of scenario graphs. It has 2 sub-steps:

a) For each requirement *ReqN*, organizing the relevant interactions into a scenario graph *SceN* to express that *SceN* can realize *ReqN*;
b) Checking the scenario graphs with the properties of the scenario graphs; if they are correct, go to the next step; otherwise, show the message about the errors and return to a) to ask for re-construction.

Step 4: Performing the problem projection. This step is for decomposing problem into simpler sub-problems. And the result is a set of sub-problem diagrams. It has 2 sub-steps:

a) For each scenario, checking whether it is a well-formed scenario; if it is, go to the next step; otherwise, go back to step 1a) for more information with strategies in [3];
b) Performing the problem projection according to the operator of projection upon well-formed scenario in [3].

II. ARCHITECTURE AND IMPLETATION

The architecture of DPTool is shown in Fig. 2. It includes 4 layers. They are the interface layer, the functional description layer, the knowledge representation layer, and the data storage layer. The knowledge representation layer provides the

1090-705X/10 $26.00 © 2010 IEEE
DOI 10.1109/RE.2010.58

basic elements representation in problem description for the functional description layer. The functional description layer supplies data to the data storage layer and provides functional supporting for the interface layer. And the interface layer provides information for the functional description layer.

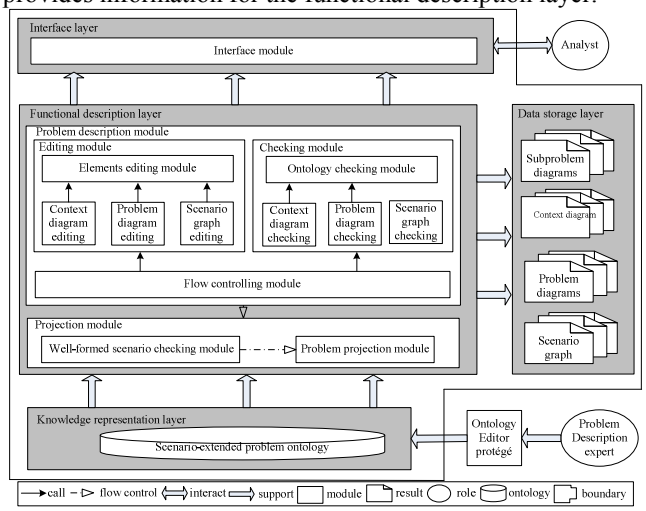

Figure 2. Architecture of DPTool

A. Kowledge Representation Layer

The knowledge representation layer provides knowledge about problem description, including basic concepts about problem description, associations among these concepts, and constraints on these concepts and associations. The scenario-extended problem ontology is responsible for this. It is built by problem description expert with protégé.

B. Functional Description Layer

The functional description layer is responsible for problem description and projection. It includes the *problem description module* and the *projection module*.

The problem description module includes the *editing module*, the *checking module*, and the *flow controlling module*. The ***editing module*** includes the *elements editing module*, the *context diagram editing module*, the *problem diagram editing module*, and the *scenario graph editing module*. Where, the *context diagram editing module* is in charge of editing instances of machine, domains and interactions according to the scenario-extended problem ontology. The *problem diagram editing module* is in charge of editing instances of requirement and references according to the scenario-extended problem ontology. *The scenario graph editing module* is in charge of editing instances of scenario graphs according to syntax of the scenario. *The elements editing module* is in charge of the basic elements in the context diagram, the problem diagram and the scenario graphs.

The ***checking module*** includes the *context diagram checking module*, the *problem diagram checking module*, the *scenario graph checking module* and the *ontology checking module*. The *context diagram checking module* is in charge of checking three elements in context diagram according to the constraints provided by the ontology. The *problem diagram checking module* is in charge of checking two elements in problem diagram according to the constraints in the ontology. The *scenario graph checking module* is in charge of checking scenario with the syntax of scenario. The *ontology checking module* is in charge of connecting ontology with the context diagram and the problem diagram checking. The ***flow controlling module*** is in charge of invoking the functions according to the process in Fig. 2.

The projection module includes the *well-formed scenario checking module* and the *problem projection module*. The *well-formed scenario checking module* is in charge of checking whether the scenario is well-formed scenario to ensure the result of projection is an integrated sub-problem. The *problem projection module* is in charge of performing problem projection operations defined in [3].

C. Data Storage Layer

The data storage layer stores the intermediate and final result of the problem description and projection. The results include context diagram, problem diagram, scenario graphs, and sub-problem diagrams.

D. Interface Layer

The interface layer is the interface for interacting with the analysts. Analysts provide information through interface.

DPTool has been implemented in Java (JDK 1.4) with IDE JBuilder 9. Fig. 3 shows the snapshot of performing the problem description and problem projection by using DPTool.

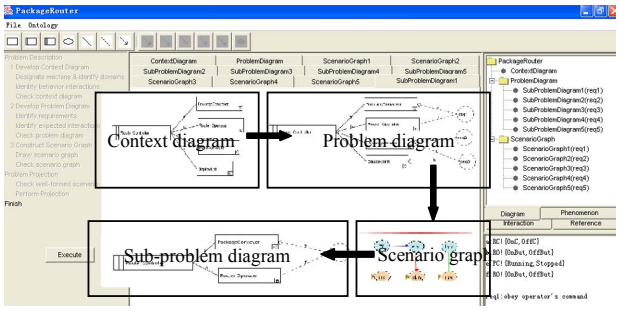

Figure 3. Snapshot of the problem description and projection

III. CONCLUSION

This paper presents a tool for guiding the problem description and projection. It graphically provides various editing, checking and verifying functions for the problem description, and supports the problem projection. Future work includes nonfunctional requirements description.

REFERENCES

[1] M. Jackson, "Problem Frames: Analyzing and Structuring Software Development Problems," Addison-Wesley, 2001.

[2] K. Cox , J.G. Hall, and L. Rapanotti, "Editorial: A Roadmap of Problem Frames Research ", Information and Software Technology, vol. 47, issue 14, pp. 891-902, 2005.

[3] Z. Jin, X. Chen, and D. Zowghi, "Performing Projection in Problem Frames Using Scenarios," Proceedings of 16th Asia-Pacific software Engineering Conference, APSEC 2009, pp.249-256, 2009.