Performing Projection in Problem Frames Using Scenarios

Zhi Jin*†, Xiaohong Chen† and Didar Zowghi‡
*Key Laboratory of High Confidence Software Technologies, Ministry of Education, Peking University, China
†Academy of Mathematics and System Science, Chinese Academy of Sciences, China
‡Faculty of Engineering & Information Technology, University of Technology, Sydney, Australia

***Abstract*—In the Problem Frames (PF) approach there are five basic problem frames and some variants to them. When a problem is being analysed, it is initially matched against these frames. If the problem does not fit into the basic problem frames or their variants, then problem analysis is performed. It has been recognised that 'projection' is an effective technique for analysing problems. That is, each sub-problem is considered as a projection of the main problem concerned only with the phenomena relevant to that sub-problem. The PF approach lacks a precise definition of problem projection and does not provide specific instructions on how to perform this projection.**

In this paper, we use the concept of projection from relational algebra and combine it with concepts from the PF and scenario-based approaches to present a conceptual model for conducting problem projection in requirements engineering. This model and ontology extend problem description at scenario level and support systematic derivation of sub-problems from scenarios. We also provide a detailed process description for performing projection for problem analysis and present the utility of our approach with a case study.

***Keywords*-Requirements Engineering; Problem Frames; Scenarios; Projection;**

I. Introduction

The PF approach [1] assumes five basic problem frames, i.e., the required behavior frame, the commanded behavior frame, the information display frame, the simple workpieces frame and the transformation frame, and some variants to them. When a problem is being analysed, it is initially matched against the basic problem frames or their variants. If the problem does not fit into either the basic problem frames or their variants, then the problem analysis is performed. It has been argued that 'projection' is an effective technique for analysing problems in the PF approach [1]. That is, each subproblem is a projection of the full problem, and concerns only the phenomena that are relevant to that subproblem. Here 'projection' has the same meaning as in relational database theory. However, the current PF approach lacks a precise definition of problem projection and does not provide specific instructions on how to perform this projection. That makes problem analysis an empirical, tedious, and subjective process, heavily dependant on the analysts' experiences.

In relational algebra, a projection is a unary operation written as $\pi_{a_1,\cdots,a_n}(R)$ where $a_1,\cdots,a_n$ is a set of attribute names. The result of such projection is defined as the set obtained when the components of tuple R are restricted to set $\{a_1,\cdots,a_n\}$. This implies that a projection operator π has two related components: the projective object, R, and the projected aspect, $\{a_1,\cdots,a_n\}$. Here, all elements in $\{a_1,\cdots,a_n\}$ are attributes in R. Corresponding to the two components in relational algebra projection, clearly in the PF approach, the projective object is the *problem*. But we need to determine what the projected aspect should be. According to Jackson [1], the essential elements of the problem description in the PF approach are: the *requirement*, the *domain properties* and the *machine specification*. Obviously, none of the above three elements in the PF approach individually are eligible to serve as the projected aspect. Therefore, we need to introduce some new element.

Scenarios have been advocated as a means of improving requirements engineering. Many interpretations of scenarios have been proposed so far. Often, scenarios are used to describe information at the instance level or examples of system behaviours [2]. They have also been claimed to be experience based narratives for requirements elicitation [3]. More precisely, scenarios are possible behaviours limited to a set of purposeful interactions taking place among participants [4].

A scenario can also be the description of system usage so that it may be considered as a pathway through a specification of system usage [5]. We argue that this particular kind of *scenario* can serve as the projected aspect in the PF approach. For this purpose, we designate a scenario to be a meaningful flow of the interactions between the machine and the problem domains. By introducing this kind of scenario, in this paper we present an integrated ontology. The elements of this ontology are concepts derived from both the PF and the scenario-based approaches. We call this ontology *the scenario-extended problem ontology*. This ontology will enable us to systematically describe the scenario-extended problem and derive the subproblems by using scenario based problem projection. Our novel extension of PF with the combination of scenarios and projection technique provides a generalisable and repeatable way for analysing and decomposing all kinds of software problem.

This paper is organized as follows. Section II presents the scenario-extended problem ontology. Section III shows how to describe the problem based on this ontology. Section IV defines the scenario based problem projection with an illustration of a case study. Section V presents related works. Finally, Section VI concludes the paper.

II. Scenario-Extended Problem Ontology

We have previously developed an ontology for PF approach [6]. Here, we use scenario related concepts [7] to extend that ontology and present a model for scenario-extended problem description. Figure 1 gives the concept categories and their associations of this ontology.

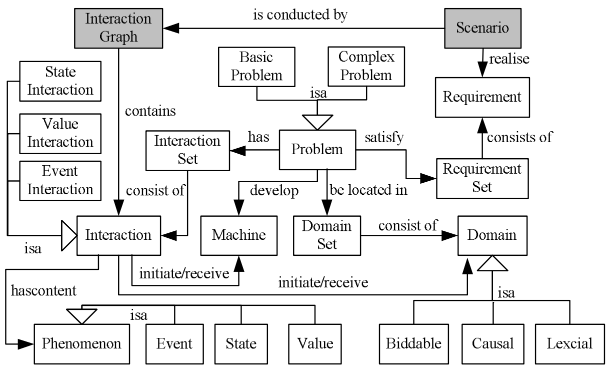

Figure 1. Conceptual Model of Problem Description

The white boxes in Figure 1 represent the concept categories in the PF approach. They capture the main principles of the PF approach: *Problem* is located in a set of real world *Domains* and is to develop a *Machine* to satisfy *Requirements*. There are shared phenomena among the *Machine* and the *Domains*, and they are *Interactions* between the Machine and the Domains. A *Domain* can be *Biddable*, *Lexical* or *Causal*. An *Interaction* has one initiator and one receiver that could be a machine or a domain and can be of *Event*, *State* or *Value*. Each interaction represents one individual action that the machine is involved in.

In the PF approach, nothing is said about how interactions and requirements are related to one another. We introduce scenarios to express how interactions can satisfy the requirements; thus addressing this weakness in the PF approach. A scenario is a flow of interactions in which each node is an interaction between the machine and a domain; and each edge corresponds to the order of two interactions.

Table I gives the meanings of these concept categories. Apart from the concept categories and the conceptual structure, some constraints on the concept categories need to be specified for imposing conditions that have to be satisfied in problem description. Table II presents the assertions of some constraints. They are self-explanatory when assuming the notations.

III. Guided Process for Scenario-Extended Problem Description

The above ontology serves as a model for the scenario-extended problem description. As illustrated in the ontology, a *Problem* associates with four concept categories. They are a *Machine*, a *Domain Set*, an *Interaction Set* and a

Table I
Hierarchy of the Concept Categories

Concept Category	**Meaning**
Problem	*a problem is to create a machine that will serve useful purpose. It refers to the domains that the machine will interact with and the requirements to be satisfied through the shared phenomena between the machine and the domains*
Basic Problem	*a kind of problem pattern which defines an identifiable problem class in terms of its interactions and the characteristics of its domain. It has a single functionality as its requirement. Problem frames are instances of basic problem*
Complex Problem	*a problem which can not fit to a problem pattern. It normally has multiple functionalities and can be composition of problem frames*
Machine	*the system to be built in a problem*
Domain	*a relevant real world entity that will interact with the machine*
Biddable Domain	*a biddable domain is one which is physical but lacks positive predicable internal causality*
Causal Domain	*a causal domain is one whose properties include predictable causal relationship among its shared phenomena*
Lexical Domain	*a lexical domain is one which is a physical representation of data*
Domain Set	*a set of domains*
Requirement Set	*a set of requirements*
Requirement	*functionalities which need to be satisfied*
Interaction Set	*A set of interactions*
Interaction	*an interaction is an observable shared phenomena between machine and a domain*
Event Interaction	*an event interaction shares an event*
Value Interaction	*a value interaction shares a value*
State Interaction	*a state interaction shares a state*
Interaction Graph	*an interaction graph is a direct graph. Its nodes are interactions and the edges express the order relationships between two interactions*
Phenomenon	*a phenomena can be of state, value and event*
Event	*an event is an individual happening. Each event is indivisible and instantaneous*
Value	*a value is an intangible individual that is not subject to change*
State	*a state is a relation among causal domains and values. It can change over time*
Scenario	*a scenario is an indivisible interaction graph that realise a requirement*

Table II
Constraints on Concept Categories and Associations

Notations:
x, y, z, $\cdots$: variables for concept instances
ConceptCategory(x): x is an instance of 'Concept Category'
Association(x,y): x and y are two concept instances, and there is an 'Association' link between them

Interaction(x),initiator(x,y),receiver(x,z)$\rightarrow$ (Domain(y)$\wedge$Machine(z))$\vee$(Domain(z)$\wedge$Machine(y))
StateInteraction(x),initiator(x,y)$\rightarrow$CausalDomain(y)
ValueInteraction(x),initiator(x,y)$\rightarrow$LexicalDomain(y)
EventInteraction(x),initiator(x,y)$\rightarrow$ BiddableDomain(y)$\vee$Machine(y)
StateInteraction(x),content(x,z)$\rightarrow$State(z)
ValueInteraction(x),content(x,z)$\rightarrow$Value(z)
EventInteraction(x),cntent(x,z)$\rightarrow$Event(z)

Requirement Set. Thus, identifying a problem is equivalent to identifying a 4-tuple $P := < M, DS, IS, RS >$ where:

- M(Machine) is the to-be-built machine
- DS(Domain Set) is a finite set of domains that will interact with the machine
- IS(Interaction Set) is a finite set of interactions in the direct interface of the machine and its domains
- RS(Requirement Set) is a set of requirements to be satisfied by the machine and its domains

Furthermore, as argued above, this problem description lacks the associations between the requirement and the machine's behaviours (i.e. the interactions that the machine is involved in). So, besides the four parts, the problem description needs one more part, i.e. the scenarios for relating the requirements to the interactions. In fact, including scenarios is mainly for justifying if the requirements in RS can be realised by the machine's behaviours. According to Section II, a scenario is a flow of the interactions, i.e. an interaction graph. Roughly speaking, an interaction graph is like a workflow except that the nodes are interactions rather than actions. The link between a scenario and a requirement captures the meaning of realising the requirement by the process of the interactions prescribed in the interaction graph. Thus, to allow problem description and analysis, the next step is to identify and prescribe the scenarios for attaining the requirements. Using practical techniques from [8], the process of describing a problem is in 6 steps:

Step 1 Assigning a name $MacN$ to the to-be-built machine. That leads to assertion:

$$Machine(MacN, Description)$$

Step 2 Identifying domains, $DomN_1$, $\cdots$, and $DomN_{n_{dom}}$, that will interact with the machine. Then

$$DS = \{DomN_1, \cdots, DomN_{n_{dom}}\}$$

Each domain may be of different types, i.e. *causal*, *biddable* or *lexical*. Any domain is of at least one type. That leads to assertions $(1 \leq i \leq n_{dom})$:

$$Domain(DomN_i, Description, DomTypeSet)$$

Step 3 Identifying the shared phenomena, $PheN_1$, $\cdots$, and $PheN_{n_{phe}}$, that are in the direct interfaces between machine and the domains. Each of these phenomena is of one and only one type, i.e. $PheType$ can be *event*, *state* or *value*. That leads to assertions $(1 \leq i \leq n_{phe})$:

$$Phenomenon(PheN_i, Description, PheType)$$

Step 4 Identifying the interactions, $IntN_1$, $\cdots$, and $IntN_{n_{int}}$, between the machine and the domains. Then

$$IS = \{IntN_1, \cdots, IntN_{n_{int}}\}$$

Each of them is represented by an assertion which is formed as $(1 \leq i \leq n_{int})$

$$Interaction(IntN_i, Ini, Rec, Phe)$$

That means that $IntN_i$ is an interaction, its initiator is Ini, its receiver is Rec, and its content is Phe which is an identified shared phenomenon.

Step 5 Identifying the requirements, $ReqN_1$, $\cdots$, and $ReqN_{n_{req}}$, that are desired to be satisfied by the problem. Recording

$$RS = \{ReqN_1, \cdots, ReqN_{n_{req}}\}$$

For each $ReqN_i$ $(1 \leq i \leq n_{req})$, identifying the shared phenomena referred to by this requirement. The shared phenomena referred by a requirement are requirement references that will be of normal references or constraining references. The former means that the requirement refers to the interaction (as-is interaction) and the latter means that the requirements asks the interaction (to-be interaction). Assertion

$$Reference(ReqN_i, Ini, Rec, Phe, RefType)$$

is used for representing the shared phenomena referred by $ReqN_i$. $RefType$ can be *normal* or *constraining*.

Step 6 For each requirement $ReqN \in RS$, organising the relevant interactions into an interaction graph to describe a scenario $SceN =< IntSet, AssSet >$ so that $SceN$ can realise $ReqN$. Here, $IntSet \subseteq IS$ contains two subsets of nodes: one is for grouping the domain referred interactions (denoted by $DomInt$) and the other is for grouping the requirement referred interactions (denoted by $ReqInt$). And $AssSet = \{assType(Int_1, Int_2)|Int_1, Int_2 \in IntSet\}$, in which, $assType$ is of the following five association types:

- $behOrd : DomInt \rightarrow DomInt$: The order of the domain referred behaviours
- $reqOrd : ReqInt \rightarrow ReqInt$: The order of the requirement references
- $behEna : DomInt \rightarrow ReqInt$: The domain referred behaviour enables the requirement reference
- $reqEna : ReqInt \rightarrow DomInt$: The requirement reference enables the domain referred behaviour
- $syncBehReq : ReqInt \leftrightarrow DomInt$: The synchronicity of the domain referred behaviour and the requirement reference

After specifying the scenarios, asserting

$$Realisation(SceN, ReqN)$$

to denote that $SceN$ realises $ReqN$.

Here, we provide an example, the package router problem [1], [9], for illustrating the process. We use Jackson's abbreviations [1] to denote the modeling elements for ease of understanding. The context diagram of this problem is omitted due to space limitation but it can be found in [1].

"*A **package router** is a large mechanical device used by postal and delivery organizations to sort packages into bins according to their destinations. The packages carry bar-coded labels. They move along a **conveyor** to a reading station where their package-ids and destinations are read. They then slide down pipes fitted with sensors at top and bottom. The pipes are connected by two-position switches that the computer can flip. At the leaves of the tree of pipes are destination bins, corresponding to the bar-coded destinations.*

A misrouted package may be routed to any bin, an appropriate ***message being displayed****. There are control buttons by which an* ***operator*** *can command the controlling computer to stop and start the conveyor.*

The problem is to build the controlling computer (1) ***to obey the operator's commands****, (2)* ***to route packages to their destination bins*** *by setting the switches appropriately, and (3)* ***to report misrouted packages****.*"

In **Step 1**, a name is given to the machine. The name can be *Router Controller (RC)*. So asserting

Machine(man_1, Router Controller (RC))

In **Step 2**, from the context diagram, four domains can be identified. They are (dom_1)*Package Conveyor (PC)*, (dom_2)*Router & Packages (RP)*, (dom_3)*Display Unit (DU)* and (dom_4)*Router Operator (RO)*. Recording $DS = \{dom_1, dom_2, dom_3, dom_4\}$. Among the four domains, *RC* and *DU* are *causal*, *RP* is *causal* and *lexical*, while *RO* is *biddable*, so the following assertions are obtained:

Domain(dom_1, PC, {*causal*})
Domain(dom_2, RP, {*causal*, *lexical*})
Domain(dom_3, DU, {*causal*})
Domain(dom_4, RO, {*biddable*})

In **Step 3**, the shared phenomena between machine and the domains are identified. For example, examining *Package Conveyor (PC)*, the shared phenomena between PC and RC are OnC and $OffC$ events by which the machine can start and stop the conveyor. Doing the same for the other domains leads to assertions:

Phenomenon(phe_1, *OnC*, {*event*})
Phenomenon(phe_2, *OffC*, {*event*})
Phenomenon(phe_3, *ShowPkgId*, {*event*})
Phenomenon(phe_4, *ShowBin*, {*event*})
Phenomenon(phe_5, *ShowDestn*, {*event*})
Phenomenon(phe_6, *LSw(i)*, {*event*})
Phenomenon(phe_7, *RSw(i)*, {*event*})
Phenomenon(phe_8, *SendLabel(p,l)*, {*event*})
Phenomenon(phe_9, *LId(l,i)*, {*event*})
Phenomenon(phe_{10}, *LDest(l,d)*, {*event*})
Phenomenon(phe_{11}, *SwPos(i)*, {*state*})
Phenomenon(phe_{12}, *SensOn(i)*, {*state*})
Phenomenon(phe_{13}, *OnBut*, {*event*})
Phenomenon(phe_{14}, *OffBut*, {*event*})

In **Step 4**, these phenomena are assigned to the direct interfaces between the machine and the domains for representing the interactions. They are:

Interaction(int_1, RC, PC, *Onc*)
Interaction(int_2, RC, PC, *OffC*)
Interaction(int_3, RC, DU, *ShowPkgId*)
Interaction(int_4, RC, DU, *ShowBin*)
Interaction(int_5, RC, DU, *ShowDestn*)
Interaction(int_6, RC, RP, *LSw(i)*)
Interaction(int_7, RC, RP, *Rsw(i)*)
Interaction(int_8, RP, RC, *SendLable(p,l)*)
Interaction(int_9, RP, RC, *LId(l,i)*)
Interaction(int_{10}, RP, RC, *LDest(l,d)*)
Interaction(int_{11}, RP, RC, *SwPos(i)*)
Interaction(int_{12}, RP, RC, *SensOn(i)*)
Interaction(int_{13}, RO, RC, *OnBut*)
Interaction(int_{14}, RO, RC, *OffBut*)

In **Step 5**, three requirements are recognised. For man_1, req_1 is the requirement that includes three sub-requirements: (req_2) *Obeying the Operator's Commands*, (req_3) *Routing Packages* and (req_4) *Reporting Misrouted Packages*. So, $RS = \{req_1\} = \{req_2, req_3, req_4\}$.

Then, the shared phenomena referred by each requirement are identified. Let us use req_2 as an example. Obviously, it refers to RO's event phenomena, *OnBut* and *OffBut*, and constrains PC's state phenomena, *Running* and *Stopped*, so the following assertions can be obtained:

Reference(req_2, RO, RC, *OnBut*, *normal*)
Reference(req_2, RO, RC, *OffBut*, *normal*)
Reference(req_2, PC, RC, *Running*, *constraining*)
Reference(req_2, PC, RC, *Stopped*, *constraining*)

Here, two more phenomena are revealed that are shared by PC and RC, so do the interactions:

Phenomenon(phe_{15}, *Running*, {*state*})
Phenomenon(phe_{16}, *Stopped*, {*state*})
Interaction(int_{15}, PC, RC, *Running*)
Interaction(int_{16}, PC, RC, *Stopped*)

In the same way, the rest of the requirement references can be obtained:

Reference(req_3, RP, RC, *PkgArr(p,b)*, *constraining*)
Reference(req_3, RP, RC, *Assoc(d,b)*, *constraining*)
Reference(req_3, RP, RC, *PDest(p,d)*, *constraining*)
Reference(req_4, RP, RC, *PkgArr(p,b)*, *normal*)
Reference(req_4, RP, RC, *Assoc(d,b)*, *normal*)
Reference(req_4, RP, RC, *PDest(p,d)*, *normal*)
Reference(req_4, RC, DU, *ShowPkgId*, *constraining*)
Reference(req_4, RC, DU, *ShowBin*, *constraining*)
Reference(req_4, RC, DU, *ShowDestn*, *constraining*)

And some more shared phenomena and interactions are:

Phenomenon(phe_{17}, *PkgArr(p,b)*, {*event*})
Phenomenon(phe_{18}, *Assoc(d,b)*, {*event*,*value*})
Phenomenon(phe_{19}, *PDest(p,d)*, {*event*})
Interaction(int_{17}, RP, RC, *PkgArr(p,b)*)
Interaction(int_{18}, RP, RC, *Assoc(d,b)*)
Interaction(int_{19}, RP, RC, *PDest(p,d)*)

Step 6 is for organising interactions into scenarios to realise requirements. Current recognised requirements are used as the thread for organising scenarios. Thus, three scenarios are captured corresponding to the three requirements as shown in Figure 2, so the three assertions are:

Realisation(sce_1, req_2)
Realisation(sce_2, req_3)
Realisation(sce_3, req_4)

IV. Scenario-based Problem Projection

Realistic problems are composite. They must be decomposed into simple subproblems that can match the basic

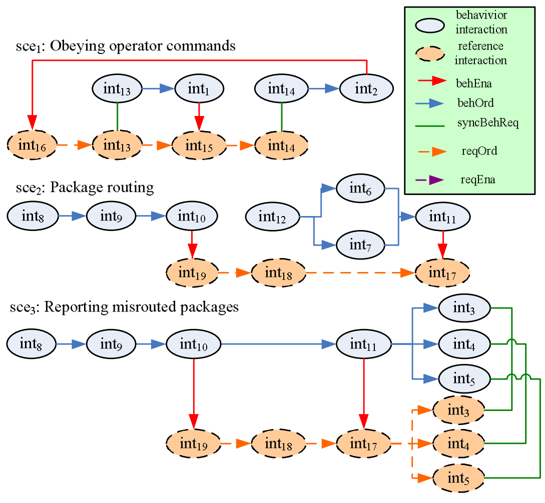

Figure 2. Package Router Control Problem: Scenarios

problem frames or their variants so that the analyst can follow the frame concerns to explore the subproblems and develop the specifications. When the problem fits into a basic problem frame, this problem frame will provide an effective and systematic approach for problem analysis. However, problem decomposition is not straight forward, and Jackson suggests to consider the subproblems as the projections of the problem [1]. This would allow the subproblems to be overlapped in some way so that the information about one domain can be distributed over several subproblems. On the other hand, because of the vague and overlapped boundaries we may not be able to systematically and accurately carry out decomposition to obtain the precise descriptions of the subproblems. For each subproblem we need to precisely describe all of the particular elements, i.e. the machines, the domains, the shared phenomena and the requirements.

This section explores the problem projection. We first recall what projection is in relational algebra that can be formally defined as follows:

$$\pi_{a_1,\cdots,a_n}(R) = \{t[a_1,\cdots,a_n] | t \in R\}$$

where $t[a_1,\cdots,a_n]$ is the restriction of the tuple t to the set $\{a_1,\cdots,a_n\}$ so that

$$t[a_1,\cdots,a_n] = \{(a',v) | (a',v) \in t, a' \in \{a_1,\cdots,a_n\}\}$$

The result of a projection $\pi_{a_1,\cdots,a_n}(R)$ is defined only if $\{a_1,\cdots,a_n\}$ is a subset of the header of R.

The problem projection takes similar form. As stated in Section I, the purpose of the problem projection is to identify subproblems in terms of scenarios. So the formation of the problem projection will be:

$$\pi^{req}_{sce}(\mathbf{P}) = \mathbf{SP}$$

Here, **P** is a problem, sce is a scenario for realising requirement req in **P**, then **SP** is a subproblem of **P** such that sce realises req.

Before the projection operator can be defined, some preliminary concepts need to be introduced. Firstly, a problem domain can have different characteristics in a problem. It can be dynamic and/or static [1].

Definition IV.1 *Let* $\boldsymbol{P} =< \mathrm{M}, DS, IS, RS >$ *be a problem,* $d \in DS$ *an initiator or a receiver in interaction* $int \in IS$*. Then* d *is dynamic in* $\boldsymbol{P}$ *if* int *is an event interaction or a state interaction.* d *is static in* P *if* int *is a value interaction.*

Secondly, the concept 'well-formed scenario' is needed when defining the problem projection as we need a meaningful scenario rather than a random interaction flow.

Definition IV.2 *Let* $\boldsymbol{P} :=< \mathrm{M}, DS, IS, RS >$ *be a problem,* $sce =< IntSet, AssSet >$ *a scenario for* $\boldsymbol{P}$ *and* $IntSet = DomInt \cup ReqInt$*. Assume* $\rho(sce)$ *returns the set of domains involved in sce. Then sce is well-formed if*

- *All* $d \in \rho(sce)$ *is either dynamic or static, but not both*
- *Let* $intR \in ReqInt$ *and* $intB \in DomInt$
  - *If* $intR$ *is an event or value interaction, then there exists behaviour enabling association* $behEna(intR, intB)$
  - *If* $intR$ *is a state interaction, then there exists requirement/behaviour synchronicity association* $syncBehReq(intB, intR)$
  - *If* $intB$ *is a machine controlled event or value interaction, then there exists requirement enabling association* $reqEna(intB, intR)$

Finally, some auxiliary projection operators are defined:

- The projection of a machine upon a scenario for realsing a requirement is one of its sub-machines $\pi^{req}_{sce}(\mathrm{M}) =$ M' so that M' is a sub-machine of M
- The projection of a domain set upon a scenario for realising a requirement is a domain set which contains all of the domains involved in this scenario: $\pi^{req}_{sce}(DS) = \{d | (d \in DS) \wedge (d = initiator(int) \vee d = receiver(int)) \wedge (int \in Nodes(sce))\}$
- The projection of an interaction set upon a scenario corresponding to a machine is an interaction set which contains all the interactions in this scenario: $\pi^{req}_{sce}(IS) = \{int | (int \in IS) \wedge (int \in Nodes(sce))\}$
- The projection of a requirement set upon a scenario corresponding to a machine is a requirement set which contains all the requirements realised by this scenario: $\pi^{req}_{sce}(RS) = \{req | req \in RS\}$

Projection upon Well-Formed Scenario When a well-formed scenario is constructed, the problem projection can be performed: Let $\mathbf{P} =< \mathrm{M}, DS, IS, RS >$ be the problem description, and sce a well-formed scenario for **P** for realising req. Then a subproblem **SP** of problem **P** upon sce for req (i.e subProblem(**SP**,**P**,sce,req)) can be obtained by

$$\mathbf{SP} = \pi^{req}_{sce}(\mathbf{P}) =< \pi^{req}_{sce}(\mathrm{M}), \pi^{req}_{sce}(DS), \pi^{req}_{sce}(IS), \pi^{req}_{sce}(RS) >$$

For example, in the package router problem, let the problem description be $< M, DS, IS, RS >$ and sce_1, sce_2 and sce_3 three scenarios to realise req_2, req_3 and req_4 respectively. It is easy to justify that sce_1 for realising req_2 is well-formed. So the subproblem upon sce_1 is

$$\pi^{req_2}_{sce_1}(PackageRouterProblem) =< M_2, DS_2, IS_2, RS_2 >$$

where

- $M_2 = \pi^{req_2}_{sec_1}(RouterController) = CommandObeyer$ is the machine name of this subproblem
- $DS_2 = \pi^{req_2}_{sce_1}(DS) = \{\text{PC}, \text{RO}\}$
- $IS_2 = \pi^{req_2}_{sce_1}(IS) = \{int_1, int_2, int_{13}, \cdots, int_{16}\}$
- $RS_2 = \pi^{req_2}_{sce_1}(RS) = \{req_2\}$

If the scenarios are not well-formed, some new domains or new problems need to be introduced.

(Strategy I) Elaborating the Problem by Introducing Model Domain A problem may contain a real world domain that is both dynamic and static. In this case, a model domain needs to be introduced that corresponds by analogy to a real world domain for separating the two concerns. Here, the model domain is a mapping of the original real world domain. The model domain will be engaged in the place where a static domain is needed while the real world domain is used where a dynamic domain is needed. Thus, the problem can be split into two subproblems: one builds the model (in dynamic way) and the other uses it (in static way).

For example in the package routing problem, the package & router is a domain of both dynamic and static type. It is static because it is a router layout for a particular package so that the package can go to the right bin. It is dynamic because the packages of different destinations may lead to different router layouts triggered by the events, $LSw(i)$ and $RSw(i)$. In this case, a router layout model needs to be included for separating the two concerns.

When a model domain $\sigma(d)$ ($\sigma(d)$ represents the model domain of d) is introduced to separate the two concerns of the same domain, the problem description $\mathbf{P} =< M, DS, IS, RS >$ can be elaborated using the following steps:

Step 1. Introducing a model domain $\sigma(d)$ into DS; $DS = DS \cup \{\sigma(d)\}$. Then, two strategies can be adopted for accomplishing the model building. One is introducing a biddable domain as the model builder, which results in another new domain to be added in DS. The other is developing an automatic model builder whose functionality needs to be included in the model building requirement.

Step 2. Identify new phenomena and interactions involving $\sigma(d)$ and other new domains. Add them into IS; Change IS's original phenomena and interactions so that those two concerns are separated because of the introduction of $\sigma(d)$.

Step 3. Split the original requirements into two separated sub-requirements. One is for building $\sigma(d)$ and the other for using $\sigma(d)$ to realise the rest of the functionality of the original requirements.

Step 4. The elaboration results in scenario separation. After that, the original scenario can be elaborated accordingly and projection can be performed.

(Strategy II) Elaborating the Problem by Introducing New Problems Sometimes, the scenario contains some undetermined interactions, e.g. missing the behaviour enabling, the requirement enabling, or the behaviour/requirement synchronicity associations. That means that there are gaps among the machine's behaviours, the domain properties and the requirements. In this case, some new problems need to be introduced to allow the missing associations to hold.

Elaborating problem by introducing new problems also demands eliciting further domains, phenomena, interactions and requirements in terms of the conceptual model, and then constructing new scenarios and elaborating the original scenarios in accordance to the elaborated problem.

Referring to Figure 2 for illustration. sce_3 is not well-formed as the requirement reference interaction int_{18} misses a requirement enabling association which can enable the correspondence between destination d and bin b. According to Strategy II, a new problem is needed. One way is to allow a 'Destination Informant (DI)' to assign the association, i.e. build 'Destination-Bin Mapping (DBM)'. This problem is developing a machine 'Destination Editor (DE)' for creating and editing DBM by following the process of problem description. The following assertions can be obtained and sce_4 can be constructed as shown in Figure 3.

Machine(man_5, 'Destination Editor (DE)')
Domain(dom_5,DI,{*biddable*})
Domain(dom_6,DBM,{*lexical*})
Phenomenon(phe_{20}, *Edit Commands*, {*event*})
Phenomenon(phe_{21}, *Mapping Operations*, {*event*})
Interaction(int_{20}, DI, DE, *Edit Commands*)
Interaction(int_{21}, DE, DBM, *Mapping Operations*)
Requirement(req_5, 'destination editing')
Reference(req_5, DI, DE, *Edit Commands*, *normal*)
Reference(req_5, DBM, DE, *Assoc(d,b)*, *constraining*)
Phenomenon(phe_{22}, *Assoc(d,b)*, {*value*})
Interaction(int_{22}, DBM, DE, *Assoc(d,b)*)
Scenario(sce_4, 'editing destination-bin mapping')
Realisation(sce_4, req_5)

With the DE, the original 'reporting misrouted package' changes accordingly to be req_6 as follows so that the machine interaction int_{23} which will be inserted in between int_{10} and int_{12} can enable requirement reference interaction int_{18}. That makes sce_3 updated to be a well-formed scenario sce_6 in Figure 3, so that the problem projection can be performed.

Requirement(req_6,'reporting misrouted package')
Phenomenon(phe_{18}, *Assoc(d,b)*, {*event*})
Interaction(int_{18}, RP, RC, *Assoc(d,b)*)
Phenomenon(phe_{23}, *Assoc(d,b)*, {*value*})
Interaction(int_{23}, DBM, RC, *Assoc(d,b)*)
Scenario(sce_6, 'reporting midrouted package')
Realisation(sce_6,req_6)

Here is another example. Scenario sce_2 in Figure 2 again is not well-formed. First, the same situation for determining *Assoc(d,b)* as in sce_3 leads to also the 'destination editor' problem according to Strategy II. This problem is shared in these two scenarios.

Second, two interactions int_6 and int_7 (which are used to set the switches for correctly routing the packages) miss requirement enabling associations. In other words, without knowledge of the layout of the router, the machine can not know how to get the correct setting, and thus can not know how to make the switch setting events. In this case, again according to Strategy II, a new problem has to be included to obtain the correct setting. With this new machine, 'package & router(PR)' becomes a domain of both dynamic and static types. According to Strategy I, a model domain $\sigma(PR)$ is needed for modelling the layout of the router so that the machine can pre-calculate the correct path based on the model. In the following, we choose the simplest solution that is, allowing a layout informant edit the layout model, i.e. the 'layout informant (LI)' designates the layout (the 'router layout model (RLM)') for each pair of destination and bin via 'static layout model editor (SLME)'.

Then, the inclusion of the new model domain leads to the updating of the set of phenomena and the set of interactions, and the updating of the requirements and the scenarios accordingly with the Step 2-4 of Strategy I.

Machine(man_8, 'Static Layout Model Editor (SLME)')
Domain(dom_7,LI,{*biddable*})
Domain(dom_8,RLM,{*lexical*})
Phenomenon(phe_{24}, *Edit Commands*, {*event*})
Phenomenon(phe_{25}, *Layout Model Operations*, {*event*})
Phenomenon(phe_{26}, *Router Layout States*, {*state*})
Interaction(int_{24}, LI, SLME, *Edit Commands*)
Interaction(int_{25}, SLME, RLM, *Layout Model Operations*)
Interaction(int_{26}, PR, LI, *Router Layout States*)
Requirement(req_8, 'router layout modelling')
Reference(req_8, PR, LI, *Router Layout States*, *normal*)
Reference(req_8, RLM, SLME, *Layout Model States*, *constraining*)
Phenomenon(phe_{27}, *Layout Model States*, {*state*})
Interaction(int_{27}, RLM, SLME, *Layout Model States*)
Scenario(sce_5, 'modelling router layout')
Realisation(sce_5, req_8)

Finally, the resulting problem projection is obtained as shown in Figure 4. Here, same shaded boxes with same box labels represent the same machine or the same problem domains. They are shared by different subproblems, so a problem hierarchy can be obtained.

V. Related Work

The PF approach have attracted much attentions in recent years. Problem Oriented Software Engineering has been presented in [9] that aims at bringing both the nonformal and formal aspects of software development together in a single framework. The development is based on an explicit representation of the problem, its parts, and their systematic transformation under a formal calculus.

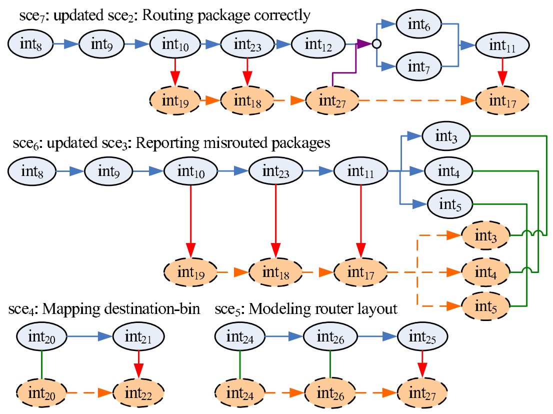

Figure 3. Package Router Control Problem: Updated and New Scenarios

Projection as a technique for problem analysis has been discussed in Jackson's book [1]. However, Jackson does not provide a precise definition of the problem projection. Furthermore, specific instructions on how to perform projection is lacking. Our paper fills this gap by providing detailed guidelines on how to perform projection in requirements engineering.

Bianco and Lavazza [10] report a preliminary investigations concerning enhancing the PF methodology with concepts derived from requirements modelling techniques based on scenarios and histories. Histories are expressed in terms of domain phenomena, and are expected to be more intuitive providing a rigorous requirements modelling framework than scenario-based modelling. However, nothing is said about projection in this work.

Haley et al [11] extend the concept of projection of a sub-problem in order to support the specification of requirements for distributed systems. They address the issues of concurrency and initialization. Our application of projection, however, goes beyond mere support for specific type of systems. The novel combination of the scenario approach with PF provides a generalisable and repeatable technique for extending PF for analysing all kinds of software problems.

VI. Conclusions

Problem decomposition is fundamental for managing problem size and complexity. This paper proposes scenario-based problem projection for analysing and decomposing complex problems. The main contributions of this paper are three folds:

- By using the concepts from the PF approach and extending it with scenarios we have developed a conceptual model for scenario-extended problem description. This ontological approach to problem description represents a novel technique for describing different classes of software problems.
- We exploit our proposed conceptual model to provide details of the problem description process. These guide-

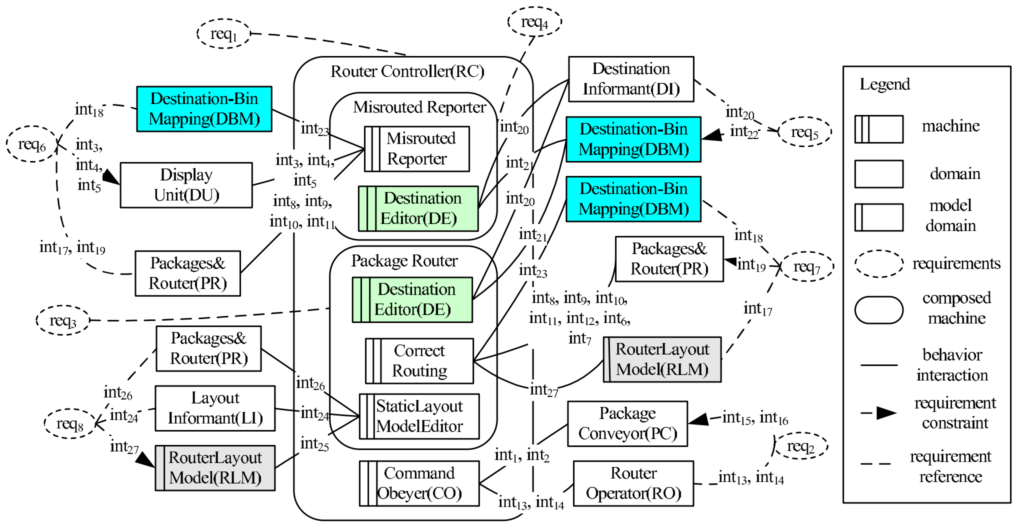

Figure 4. Package Router Control Problem: Problem Projection

lines are particularly useful for analysts during requirements elicitation as they give step-by-step instructions on how to elicit requirements though scenarios and to describe the software problems.

- We present a formal treatment of scenario-based projection with the problem frames and illustrate the rigour of our approach with a case study. This much needed formality provides potentials for automated support for problem analysis and projection.

For future works we are considering at least two immediate possibilities. Firstly, in our approach, problem projection can be performed only if the scenarios are well formed. We wish to investigate other strategies that could be used in situations where well formed scenarios are not available. Secondly, we are interested in utilising the formality of our approach in developing automated support for requirements analyst when using problem frames in RE.

Acknowledgement

This work was supported financially by the National Natural Science Fund for Distinguished Young Scholars of China (Grant No. 60625204), the National Basic Research and Development 973 Program (Grant No. 2009CB320701), the Key Projects of National Natural Science Foundation of China (Grant Nos. 90818026, 60736015), as well as the International Science Linkage Research Grant under the Australia-China Special fund for Science and Technology.

References

[1] M.Jackson, *Problem Frames: Analyzing and Structuring software development problems*. Addison-Wesley, 2001.

[2] I.Jacobson, M.Christerson, P.Jonsson, and G.Overgaard, *Object-Oriented Software Engineering: A Use-Case Driven Aproach*. Addison-Wesley, 1992.

[3] C.Potts, K.Takahashi, and A.I.Anton, "Inquiry-based requirements analysis," *IEEE Software*, vol. 11, no. 2, pp. 21–32, 1994.

[4] V.Plihon, J.Ralyté, A.Benjamen, N.A.M.Maiden, A.Sutcliffe, E.Dubois, and P.Heymans, "A reuse-oriented approach for the construction of scenario based methods," in *Proceedings of ISPA 5th International Conference on Software Process (ICSP'98)*, 1998, pp. 14–17.

[5] A.G.Sutcliffe, N.A.Maiden, S.Minocha, and D.Manuel, "Supporting scenario-based requirements engineering," *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1072–1088, 1998.

[6] X.Chen, Z.Jin, and L.Yi, "An ontology of problem frames for guiding problem frame specification," in *Proceedings of the 2nd Conference on Knowledge Science, Engineering and Management (KSEM 2007)*, 2007.

[7] C.Rolland, C.Souveyet, and C.B.Achour, "Guiding goal modeling using scenarios," *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1055–1071, 1998.

[8] B.L.Kovitz, *Practical Software Requirements: A Manual of Content and Style*. Manning Publications Co., 1999.

[9] J.G.Hall, L.Rapanotti, and M.Jackson, "Problem oriented software engineering: Solving the package router control problem," *IEEE Transactions on Software Engineering*, vol. 34, 2008.

[10] V.D.Bianco and L.Lavazza, "Enhancing problem frames with scenarios and histories: a preliminary study," in *Proceedings of the 2nd International Workshop on Advances and Applications of Problem Frames (IWAAPF 2006)*, 2006, pp. 25–31.

[11] C.B.Haley, R.C.Laney, and B.Nuseibeh, "Using problem frames and projections to analyze requirements for distributed systems," in *Proceedings of the 10th International Workshop on Requirements Engineering: Foundation for Software Quality(REFSQ'04)*, 2004, pp. 203–217.