VDM-SL in action: A FRAM-based approach to contextualise formal specifications

Tomohiro Oda[1], Shigeru Kusakabe[2], Han-Myung Chang[3], and Peter Gorm Larsen[4]

[1] Software Research Associates, Inc. (tomohiro@sra.co.jp)
[2] University of Nagasaki (kusakabe@sun.ac.jp)
[3] Nanzan University (chang@nanzan-u.ac.jp)
[4] Aarhus University, DIGIT, Department of Electrical and Computer Engineering, (pgl@ece.au.dk)

Abstract. Software development is a collaborative effort by stakeholders from different domains of expertise and knowledge. Developing a useful system needs to take different views on the system's functionalities into account from the system's internal properties, the individual user's tasks at hand, and an organisation's mission as a whole. Participation of a wide range of individual and organisational stakeholders is crucial in the specification phase. This paper proposes a use of the FRAM diagram as a pre-formal notation to describe the system's roles in individual users' tasks and organisational activities. A tool that links public operations in VDM-SL and activities in FRAM is introduced to demonstrate and discuss how a FRAM diagram contextualises a formal notation.

1 Background

Software development is often driven by multidisciplinary collaboration and the developed system is also often operated by people from different areas of expertise. The specification phase aims to explicitly define what the system to be developed shall be able to perform. A rigorous specification provides a basis to create a software system that works as planned. However, just working as planned is not enough to be useful. The plan needs to fit with various activities of stakeholders that may vary over time. Participation of a wide range of stakeholders is thus crucial to bring out an appropriate plan.

The stakeholders desire to foresee how the system to be developed will work in their activities. It is important for successful software development that the system's functionalities are understood by the stakeholders. Although a rigorous specification in a formal notation, including VDM-SL [6], concisely defines the system's functionalities without ambiguity, it is difficult for many stakeholders to understand how the specified system will work in their activities. One difficulty is the notation. Although the rapid spread of programming education may relax this difficulty, reading fluency of formal notation requires a certain time and effort. Another difficulty is contextualisation. The set of functionalities defined in the specification needs to make sense in each stakeholder's own expertise. The formal specification rigorously defines a functionality with inputs, outputs and internal states without ambiguity. However, the inputs, outputs

and internal states are defined in terms of the system's internal structure and properties although they need to be grounded and explained in the stakeholder's activities.

The third difficulty is variability. One obvious source of variability comes from the user. A user, or a group of people, sometimes performs unexpected actions on the system, or they do not perform expected actions in time. An external server also may fail to respond in time. The variability is not only from failures but also from good reasons. Data may become available earlier than expected because the person in charge of the previous action has outperformed the expectation. The data may need tentative storage until the originally expected time. Such variability emerges from actions happening out of the system, and is needed to understand how the system will work in action with foreseen variabilities.

The authors has been addressing the difficulty of notation using specification animation. Specification animation is a technique to simulate the execution of the specified system using interpreters. The stakeholders can preview the functionality of the system before the implementation without understanding the details in the formal specification. The animation-based approach, however, does not solve the difficulty of contextualisation.

In this paper, a FRAM-based approach to define and understand the contextual information of the system's functionalities. FRAM (Functional Resonance Analysis Method) will be briefly explained in Section 2. Section 3 explains existing use of FRAM in the requirement elicitation. Section 4 proposes our approach to develop formal specifications in VDM-SL with pre-formal analysis using FRAM. An example development of review bidding system will be explained in Section 5 and discussed in Section 6. Section 7 will conclude this paper.

2 Function Resonance Analysis Method

FRAM (Function Resonance Analysis Method) [3,5] was originally introduced by E. Hollnagel to unfold how a complex, dynamic and socio-technical system works and its anticipated variability in operations. The basic idea of FRAM is to identify functions or activities in a collaborative system, and analyse influences between them based on six aspects. To avoid confusion with VDM-SL's functions, we denote *activity* instead of function.

Activities can be categorised into three types according to their performers: organisational activities, human activities, and technological activities. organisational activities are those carried out by a group of people. Human activities include, but are not limited to, the end users' tasks at hand. Series of activities that directly or indirectly influences the system's activities are also identified and included in human activities. Technological activities are those performed by the developing system and also existing systems. The difference among the three types is often apparent in their variability. Human activities often vary more significantly from their original plans than technological activities. Organisational activities may vary even further than human activities. To develop a useful system, such variabilities need to be taken into account.

Once activities are identified, each of them is analysed with regard to six aspects as follows:

1. Input: what the activity processes the input, or what starts the activity.
2. Precondition: what must be done before the activity is performed.
3. Resource: what the activity needs or consumes.
4. Time: when the activity starts and/or finishes.
5. Control: what monitors, supervises or regulates the process.
6. Output: what the activity yields.

Input, precondition, resource, time and control are aspects that may receive influences of other activities while output may give influence to others. Functional couplings among activities are identified by matching the name of aspects between the first five aspects of an activity and outputs of other activities.

For example, assume that code review is mandatory after coding and a quality control team aggregates review reports from reviews. The quality control team then supplies review advise for subsequent code reviews. In this scenario, coding, code review and quality control are identified as activities. The coding activity has an output called source code, which is also input of code review. Similarly, a review report is output of the code review activity, and it is also input of the quality control activity. The quality control activity outputs review advisory, which controls the code review activity. Fig. 1 shows a FRAM diagram that visualises the identified activities and functional couplings among them.

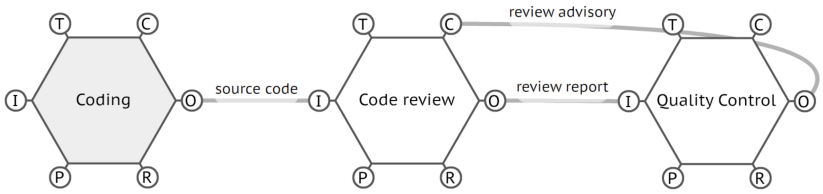

Fig. 1. An example FRAM diagram of code review

In Fig. 1, participants in the development can see that code review should be scheduled after writing source code, and may required to revise the code. For the coder, writing a code always needs code review, and scheduling a review session might be a bottleneck of coding in the long run. The quality control team does not directly discuss coding issues with the coders because they send review advise to reviewers for subsequent use.

Fig. 2 illustrates the modified code review by introducing an automated style checker driven by the Continuous Integration (CI) server. With the FRAM diagram shown in Fig. 2, the developers can see how the style checker will be used in the development project. The style checker analyses the source code and creates a report that identifies code-smells. The quality control team monitors the reports and issues review requests for suspicious code with code-smells. Please note that the review request is a precondition of the code review activity, which means that a code review session starts only if the quality control team requests it. For the coders, the style checker is not solely

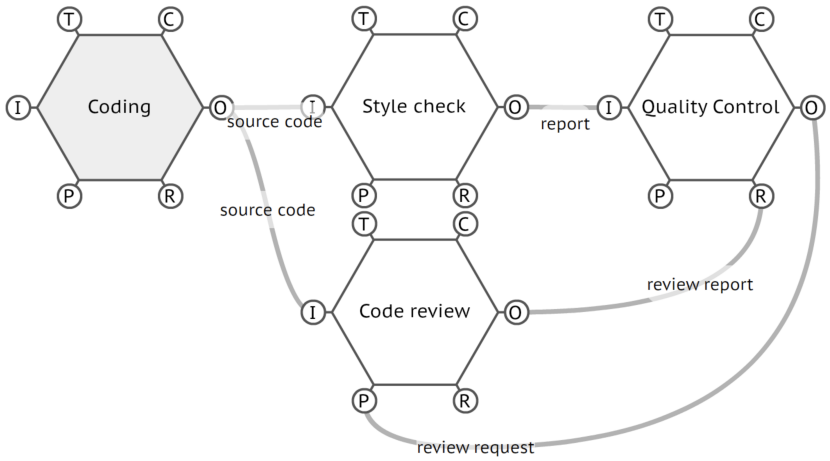

Fig. 2. An example FRAM diagram of code review with style checker

a quick code review, but rather it eliminates the delay caused by scheduling of review sessions. By introducing the style checker, the coders will not need to wait for code review sessions. The quality control team can use the style checker as a means to directly communicate with the coders through review requests.

The FRAM diagram does not provide detailed information and control structure within the collaborated activities. It instead illustrates a context of the collaborations that the system to be developed will participate in. In conjunction with formal specification, the stakeholders can preview how the system will work before the implementation. Our approach to use FRAM diagram along with VDM-SL specification will be described in the section below.

3 Related Work

De Carvalho introduced four phases to elicit functional and non-functional requirements using FRAM [2]. Experiments to elicit functional and non-functional requirements using FRAM and BPMN (Business Process Modeling Notation) were conducted, and the use of FRAM for identifying was confirmed promising. Our approach is based on the four phases performed in the experiment, namely contextualisation, aggregation, transformation, and specification.

While stakeholders requirements elicitation phase in general starts with identifying stakeholders and collecting requirements to the system, the first two phases in the experiment were to analyse how the stakeholders collaborate without the system, followed by the last two phases to address the found difficulties. In the contextualisation phase, the scope of the domain area, work processes, business goals are analysed by ethnographic observation, interviews and collecting documents. A catalog of activities in the business is produced in the contextualisation phase. In the aggregation process, the catalog

of activities is further analysed in the perspective of solutions. Functional coupling between activities and potential difficulties in the activities are identified. Transformation is the phase to produce a solution to address difficulties in the current Work-as-Done. A list of expected activities to solve the difficulties are created. In the specification phase, main features and restrictions in a natural language are identified to realise the expected activities identified in the transformation phase. The four phases go from the top to the bottom in a one-way manner.

Our approach is to produce a formal specification in VDM-SL from the FRAM diagrams created in the transformation phase. We expect the formal specification will reveal unidentified activities and hidden couplings among activities, and such feedback should be reflected to the transformation phase to form a cyclic process to develop the specification. Formal engineers can also use the FRAM diagram to produce walk-through scenarios to test the formal specification.

4 FRAM diagram to VDM-SL specification

FRAM was originally developed for analysing complex, dynamic and socio-technical systems. A FRAM diagram depicts how organisations, individuals and technological entities including computer systems work together, and its applications include requirements elicitation [2]. This paper proposes to use FRAM diagrams as a pre-formal notation to identify functionalities of the system, and specifies them in VDM-SL in a traceable manner. By *pre-formal*, we mean documents whose parts or the whole would later be specified in a formal notation. A pre-formal document could be an input to the specification phase, a reason of practical validity of the specification, or a supplemental explanation to the formal specification. Our approach takes the following steps.

1. Modeling WAD (Work as Done) in a FRAM diagram that describes how the work is currently carried out. Identify which activities need the support of a new system. If the work is not present yet, start with the next step.
2. Modeling transformation in a FRAM diagram that describes how the work will be carried out with the support of a new system. Introduce new technological activities that the new system will provide.
3. Specifying the new system in VDM-SL that corresponds to the technological activities. Use annotations to place traceability information to the FRAM diagram.
4. Providing feedback. Specify extra activities discovered or emerged in the VDM-SL specification, and place them in the FRAM diagram.

The first two steps correspond to the stakeholders requirements elicitation. While stakeholders requirements elicitation, in general, starts with collecting requirements for the system from stakeholders, the first step of our approach is to analyse how the stakeholders collaborate. Then, in the second step, the difficulties found in the first step will then be addressed by introducing a system.

The last two steps form a loop, and thus the traceability between an activity in FRAM and its counterpart in VDM-SL is important. Annotations are a mechanism to embed additional information into a VDM-SL specification[7][1]. An annotation takes the form of `--@` which is processed as a comment by interpreters by default but can also

be processed by tools that support annotations. Table 1 shows the forms of annotations that we introduced for traceability with FRAM. The annotations for FRAM traceability are implemented in ViennaTalk [8]. From a FRAM diagram, ViennaTalk can generate the annotations with a template definition of an operation for each activity. A specification with FRAM annotations can also be merged into an existing FRAM diagram as feedback.

Table 1. Annotations for traceability with FRAM

prefix	identifier	free text	description
`--@FRAM`	`Function`	*activity name*	specifies the name of a corresponding activity
`--@FRAM`	`Input`	*aspect name*	specifies the name of corresponding aspect
`--@FRAM`	`Precondition`	*aspect name*	specifies the name of corresponding aspect
`--@FRAM`	`Resource`	*aspect name*	specifies the name of corresponding aspect
`--@FRAM`	`Time`	*aspect name*	specifies the name of corresponding aspect
`--@FRAM`	`Control`	*aspect name*	specifies the name of corresponding aspect
`--@FRAM`	`Output`	*aspect name*	specifies the name of corresponding aspect

5 Example: Paper review

In this section, we describe an experimental development of a support tool for academic paper review. Assume that an academic community wants to build a system that supports the paper review. For each submitted paper, a PC chair assigns a certain number of reviewers to evaluate the quality of the paper and makes the final call to either accept or reject. Some candidate reviewers are registered. A reviewer may receive a review request from the PC chair, read the paper, and send a review report to the PC chair. If the review reports of a paper do not agree, the reviewers will start a discussion, and the PC chair is responsible to make the final call. In the rest of this section, we follow the four steps described in Section 4 to develop a formal specification of a bidding system for reviewer assignment.

5.1 Modeling WAD (Work as Done) in FRAM

Fig. 3 shows a FRAM diagram of the review process. The activity at the top row is the `Submit a paper` activity to be carried out by an author. The activities at the middle row are those carried out by the PC chairs who are responsible to assign reviewers to submitted papers, make the final calls, and notify them. The activity at the bottom row is carried out by the reviewers. The `Assign reviewers` activity requires much effort and is time consuming to find a good combination of topics of papers and the expertise of reviewers. A delay of the `Assign reviewers` activity may shorten the `Review` activity that requires enough duration to make a fair judgement to the quality of the paper. The `Make a call` activity is also difficult to complete in time when the discussion among reviewers does not converge, but less couplings with other activities are present than the `Assign reviewer` activity.

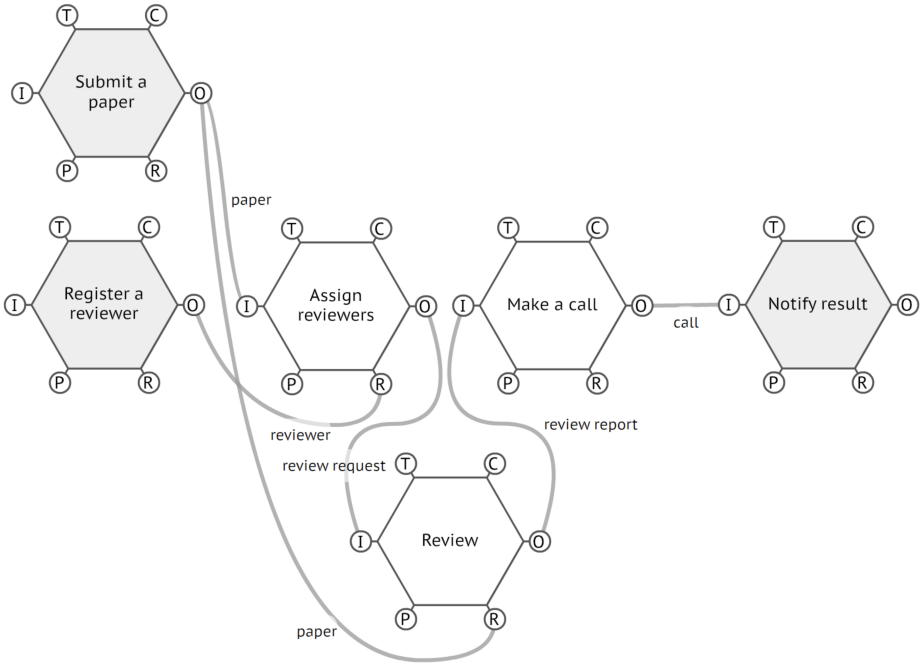

Fig. 3. An example FRAM diagram of paper review

5.2 Modeling Transformation in FRAM

The FRAM diagram shown in Fig. 4 is a solution to solve the difficulty of reviewer assignment. A bidding system is introduced to assist the reviewer assignment. Evaluating the matching between a paper and a reviewer is delegated to the reviewer. Each reviewer takes a look at submitted papers and make a *bid* to declare which paper the reviewer would be confident to review according to the reviewer's own expertise. With the bids, the system can propose to the PC chair several good combinations, and the PC chair chooses one.

5.3 Specifying the system in VDM-SL

The next step is to specify the bidding system. Fig. 5 shows the definitions of types, values and state required to define the operations listed in the FRAM diagram in Fig. 4. Each submitted paper is given a review status of either `<SUBMITTED>`, `<IN_REVIEW>`, and so on. The bidding system manages the review status of all papers in the state variable `calls`. A bid is a mapping of each paper and preference in five grades, and the preference is specified `<CONFLICT>` if the reviewer has a conflict of interests. The system holds the bids by reviewers in the state variable `bids`.

Fig.6 shows definitions of the operations that corresponds to the technological activities `Submit a paper`, `Register a reviewer`, `Bid` and `Make a call`.

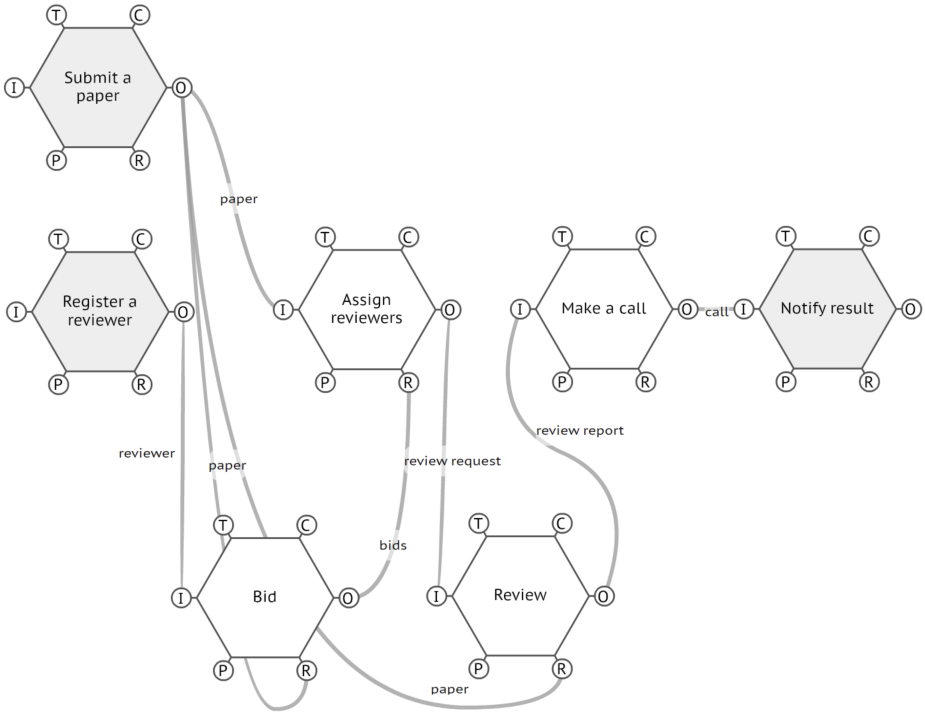

Fig. 4. An example FRAM diagram of paper review with bidding

The lines that begin with `--@FRAM` are annotations to keep traceability information with the original FRAM diagram. The operation `submit` has two annotations that tells that the operation corresponds to the `Submit a paper` activity and the activity should have an output aspect named `paper`. The contextual information in the FRAM diagram may help the specifier understand when and how the operation would be used, what inputs are given, and what effects are expected.

The FRAM diagram is drawn and analysed from the perspective of collaborative activities among organisations, people and technologies. On the other hand, the VDM specification is written from the perspective of rigorous information processing. Some extra operations may be discovered in the formal specification phase.

Fig. 7 shows the definitions of operations found possibly necessary along with the operations in Fig. 6. The `viewBid` and `viewAssignments` operations would be needed in the construction of the user interface of the `Bid` and `Assign reviewer` activities. The `viewAssignments` operation is a core part of this system to propose assignments with the highest matching to bids. These additional activities from the system's perspective should be reflected into the FRAM diagram.

Fig. 8 shows the FRAM diagram that incorporated the additional activities. One can see that a reviewer can see the own bids (the `View bids` activity) when the re-

```
types
     ID = nat;
     Reviewer = ID;
     Paper = ID;
     BidWeight = nat1 inv l == l <= 5;
     Bid = map Paper to (BidWeight| <CONFLICT>);
     Assignment = map Paper to set of Reviewer;
     Call =
        <SUBMITTED>| <IN_REVIEW>|
        <ACCEPTED>| <REJECTED>| <RETRACTED>;

values
     DEFAULT_BIDWEIGHT = 3;
     REVIEWS_PER_PAPER = 3;

state Bidding of
  calls : map Paper to Call
  bids : map Reviewer to Bid
init s ==
  s
  = mk_Bidding({|->}, {|->})
end
```

Fig. 5. Types, values and state definition for the bidding system

```
operations
  --@FRAM Function Submit a paper
  --@FRAM Output paper
  registerPaper : () ==> Paper
  registerPaper() ==
    (dcl newPaper:Paper
      := if calls = {|->} then 0 else max(dom calls) + 1;
    calls(newPaper) := <SUBMITTED>;
    bids := {r |-> ({newPaper|->DEFAULT_BIDWEIGHT} ++bids(r))
            | r in set dom bids};
    return newPaper);

  --@FRAM Function Register a reviewer
  --@FRAM Output reviewer
  registerReviewer : Reviewer ==> ()
  registerReviewer(reviewer) ==
    if
      reviewer not in set dom bids
    then
      bids(reviewer)
        := {p |-> DEFAULT_BIDWEIGHT | p in set dom calls};

  --@FRAM Function Bid
  --@FRAM Input reviewer
  --@FRAM Resource paper
  --@FRAM Output bids
  bid : Reviewer * Bid ==> ()
  bid(reviewer, bid) ==
    if
      reviewer in set dom bids
    then
      bids(reviewer)
        := dom calls <:
          ({p |-> DEFAULT_BIDWEIGHT | p in set dom calls}
           ++ bid);

  --@FRAM Function Make a call
  --@FRAM Input review report
  --@FRAM Output judgement
  changeCall : Paper * Call ==> ()
  changeCall(paper, call) == calls := calls ++ {paper |-> call};
```

Fig. 6. Operations derrived from FRAM diagram

```
--@FRAM Function View bids
--@FRAM Output bids
--@FRAM Resource bids
--@FRAM Input reviewer
viewBid : Reviewer ==> [Bid]
viewBid(reviewer) ==
  return if reviewer in set dom bids
      then bids(reviewer)
      else nil;

--@FRAM Function View assignments
--@FRAM Input bids
--@FRAM Output assignments
--@FRAM Resource paper
viewAssignments : () ==> set of Assignment
viewAssignments() ==
  (dcl
    reviewSlots:seq of [ID] := [],
    assignments:map Assignment to real := {|->},
    submitted:seq of Paper := [];
  for all p in set dom (calls :> {<SUBMITTED>}) do
    submitted := submitted ^ [p];
  while len reviewSlots < len submitted  * REVIEWS_PER_PAPER
  do for all r in set dom bids do
      reviewSlots := reviewSlots ^ [r];
  for all reviewers in set perms(reviewSlots) do
    let
      assignment : Assignment =
        {submitted(index)
        |-> {reviewers(r)
          | r
          in set {(index - 1)  * REVIEWS_PER_PAPER + 1,
          ..., index  * REVIEWS_PER_PAPER}}
        | index in set {1, ..., len submitted}},
      score = scoreByBids(assignment)
    in
      if
        score <> nil
      then
        assignments
          := assignments munion {assignment |-> score};
  return dom (assignments :> {max(rng assignments)}));
```

Fig. 7. Additional operations for bidding

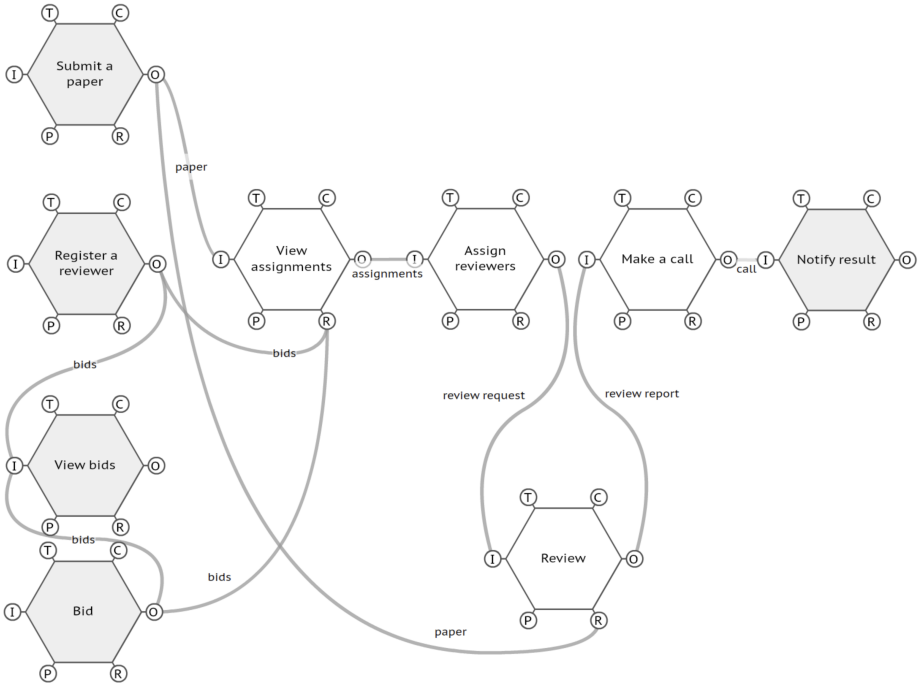

Fig. 8. An example FRAM diagram with feedback from VDM specification

viewer is newly registered (the `Register reviewer` activity by the PC chair) and also after making bids (the `Bid` activity by the reviewer). The FRAM diagram also indicates that the PC chair will see the proposed assignments of reviewers (the `View assignments` action), and then decide the assignment (the `Assign reviewers` action). The stakeholders including the research community members can see the resulting FRAM diagram in Fig. 8 to confirm whether or not the whole work flow would fit with the actual paper review process.

6 Discussions

The objective of our approach to using the FRAM diagram as a pre-formal notation is the involvement of a wider range of stakeholders. The expected use of FRAM diagrams in our approach is as an input to the specification phase, as a model to confirm the practical validity of the specification, or as a supplemental explanation to understand the formal specification. The use of the FRAM diagram is not limited to before the formal specification but also continues after the specification phase.

To support the use of FRAM diagrams in association with VDM-SL specification, a toolchain that handles both notations is required. FMV (FRAM Model Visualiser)[5]

[5] `https://zerprize.co.nz/home/FRAM`

is a commonly used diagram editor for FRAM users. FMV uses XML format to file a FRAM diagram, and thus collaboration with FMV is technically possible via files. We have extended ViennaTalk to read, modify, and write FRAM models so that technological activities in a FRAM diagram can be incorporated into a VDM model, and vice versa.

One difficulty in keeping track of associations between FRAM activities and VDM operations is vocabulary mismatching. FRAM and VDM have different scopes of the model. FRAM overviews the collaborative work as a whole while VDM specifies the functionality within a system. The difference in modelling scopes may cause mismatching of vocabularies. An activity in a FRAM diagram is worded from the perspective of the collaborative workers while its corresponding operation is named according to its functionality in the system. For example, the `Submit a paper` in Fig. 4 corresponds to the operation named `registerPaper`.

Our approach uses annotations to embed traceability information into a VDM specification. Annotations are a special form of comment intended to be processed as a comment by the standard interpreter but can place tool dependent information. Lausdahl et. al. used annotation to declare an interface with FMUs [7]. Battle et. al. used annotation as an extension to perform directives such as probing data and dispatching plugin routines at runtime [1]. We consider traceability information to be another usage of annotation not limited to FRAM but also applicable to other notations.

Our approach also introduces a cyclic process between formalisation by VDM and user feedback by FRAM. We believe such a cycle is highly encouraged especially in the early stages of the specification phase [8,9]. Fraser and Pezzoni used EAC (Executable Acceptance Criteria) to document and enforce system requirements [4]. EAC plays a key role in stakeholder involvement in Behaviour Driven Specification. We consider EAC is also documentation looking at the system's functionality from an external point of view. We expect more investigations to make more use of external views on VDM specification so that more stakeholders' interests can be reflected in the rigorous specification in VDM.

7 Concluding Remarks

Formal specification serves as a blueprint of a system's internal design and structural constructions, and hardly describes how it looks to the users and other stakeholders. The authors have been investigating the way to make formal specifications understood by a wider range of stakeholders through the development of ViennaTalk. ViennaTalk makes use of specification animation to make a formal specification understandable to the non-engineering stakeholders. The user still needs to understand the way how the system will be used. This paper focuses on FRAM as a semi-formal notation to describe the exterior views of the system in the perspective of how and why the stakeholders will use the system. We demonstrated that annotations in a VDM specification can glue the formal specification and semi-formal notations such as FRAM. Although FRAM is still in the engineering domain, its diagrammatic nature has the potential to make sense for a wider range of stakeholders. We expect further investigation including automated

generation of use scenarios from FRAM diagram and translation to VDM-SL test cases using annotations.

Acknowledgements

The authors thank Keijiro Araki for valuable and fruitful discussion on pre-formal notations. We would also like to thank anonymous reviewers for their valuable feedback. A part of this work was supported by JSPS KAKENHI Grant Number 20K11759.

References

1. Battle, N., Thule, C., Gomes, C., Macedo, H.D., Larsen, P.G.: Towards a Static Check of FMUs in VDM-SL. In: Sekerinski, E., Moreira, N., Oliveira, J.N., Ratiu, D., Guidotti, R., Farrell, M., Luckcuck, M., Marmsoler, D., Campos, J., Astarte, T., Gonnord, L., Cerone, A., Couto, L., Dongol, B., Kutrib, M., Monteiro, P., Delmas, D. (eds.) Formal Methods. FM 2019 International Workshops. pp. 272–288. Springer-Verlag, LNCS 12233 (2020)
2. de Carvalho, E.A., Gomes, J.O., Jatobá, A., da Silva, M.F., de Carvalho, P.V.R.: Employing resilience engineering in eliciting software requirements for complex systems: experiments with the functional resonance analysis method (fram). Cognition, Technology & Work 23(1), 65–83 (2021)
3. Erik, H.: FRAM: the functional resonance analysis method: modelling complex socio-technical systems. CRC Press (2017)
4. Fraser, S., Pezzoni, A.: Behaviour driven specification. In: Macedo, H.D., Thule, C., Pierce, K. (eds.) Proceedings of the 19th International Overture Workshop. Overture (10 2021)
5. Hollnagel, E.: Safety–I and safety–II: the past and future of safety management. CRC press (2018)
6. Larsen, P.G., Lausdahl, K., Battle, N., Fitzgerald, J., Wolff, S., Sahara, S., Verhoef, M., Tran-Jørgensen, P.W.V., Oda, T.: VDM-10 Language Manual. Tech. Rep. TR-001, The Overture Initiative, www.overturetool.org (April 2013)
7. Lausdahl, K., Bjerge, K., Bokhove, T., Groen, F., Larsen, P.G.: Transitioning from Crescendo to INTO-CPS. In: Fitzgerald, J., Tran-Jørgensen, P.W.V., Oda, T. (eds.) Proceedings of the 15th Overture Workshop. pp. 16–30. Newcastle University, Computing Science. Technical Report Series. CS-TR- 1513 (September 2017)
8. Oda, T., Araki, K., Larsen, P.G.: A formal modeling tool for exploratory modeling in software development. IEICE Transactions on Information and Systems 100(6), 1210–1217 (June 2017), https://www.jstage.jst.go.jp/article/transinf/E100.D/6/E100.D_2016FOP0003/_article
9. Oda, T., Yamomoto, Y., Nakakoji, K., Araki, K., Larsen, P.G.: VDM Animation for a Wider Range of Stakeholders. In: Ishikawa, F., Larsen, P.G. (eds.) Proceedings of the 13th Overture Workshop. pp. 18–32. Center for Global Research in Advanced Software Science and Engineering, National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan (June 2015), http://grace-center.jp/wp-content/uploads/2012/05/13thOverture-Proceedings.pdf, GRACE-TR-2015-06