The FLUIDE Framework for Specifying Emergency Response User Interfaces Employed to a Search and Rescue Case

Erik G. Nilsson
SINTEF ICT
Erik.G.Nilsson@sintef.no

Ketil Stølen
SINTEF ICT
Ketil.Stolen@sintef.no

ABSTRACT

The FLUIDE Framework supports development of flexible emergency response user interfaces, meeting the special challenges when developing such user interfaces. This paper presents the FLUIDE Framework with particular emphasis on its specifications languages. We demonstrate the FLUIDE Framework by giving examples from the FLUIDE specification of the user interface of an application supporting management of unmanned vehicles in search and rescue operations. We also report the findings from an experiment investigating how easy FLUIDE specifications are to understand for systems developers not knowing FLUIDE.

Keywords

User interface specification languages; SAR.

INTRODUCTION

The response to a simple everyday incident is very different from how a long-lasting serious catastrophe is handled. Increasing complexity tends to cause increasing change rate and decreasing predictability. Developing ICT solutions supporting such a range of responses is challenging. Designing user interfaces (UIs) that are able to adapt to and reflect these variations is particularly challenging.

Local leaders at the incident site should be able to use the same applications on different equipment types with different screen sizes (Nilsson and Stølen, 2010). Field workers need non-intrusive ICT support, using non-visual modalities in addition to visual ones. ICT support for emergency responders, if at all available, tends to handle the needs for flexibility by being generic and data oriented. This forces responders to adapt to the solutions, and not the other way around. There are on the other hand also many similarities and reoccurring patterns across emergency response operations. This includes tasks and information needs of individual operations and the involved actors. Previous research (Nilsson and Stølen, 2011) has shown that it is possible to characterize such similarities and patterns using a limited number of categories of functionality.

We suggest a development approach providing mechanisms for composing end-user solutions from flexible and tailor-friendly components supporting such categories of functionality. The components combine being ready-to-use with being highly configurable and support composition and configuration both at design- and run-time.

Using traditional programming languages to develop UIs for such solutions is very challenging if at all possible. Because all imaginable combinations of functionality, compositions and configurations must be supported, such UI developments are extremely resource demanding. Existing model-based UI development approaches are also seriously challenged by this such solutions. Summarized, we need for building blocks that:

R1. Are at a sufficiently high level of abstraction and provide compound structures of simple elements and containers/dialogs to support common specifications between platforms and modalities
R2. Have reflection mechanisms giving an awareness of model structures (including domain models) to support adaptation both at design- and run-time
R3. Support development of UIs where the layout depends on the instances at run-time, typically using icons, maps, graphical elements, and alternative modalities like speech
R4. Provide specific support for UI patterns that are particularly useful for emergency response

Neither MARIA (Paternò et al., 2009), UsiXML (Limbourg et al., 2004), nor OMG's Interaction Flow Modeling Language (www.ifml.org) meet R1 fully. They provide abstract building blocks, but none of the approaches offer compound building blocks. The building blocks in these approaches are abstractions of simple UI elements and containers for structuring these, but there are not any composed building blocks. With such building blocks, the composition structure of the UIs, which is different when the platforms have large differences, is reflected in the specifications even at the abstract level. MARIA and UsiXML meet R2 to some extent, and MARIA partly meets R3 by composing external UI services, while there exists extensions to UsiXML supporting maps and 3D UIs. ICOs (Navarre et al., 2009) on the other hand, meets R3 well by supporting development of post-WIMP UIs, but does not meet R1 and meets R2 only partly. R4 is best met by ICOs, which has been proven useful in domains like command and control, air traffic control and cockpit systems.

The FLUIDE Framework provides building blocks fulfilling R1-R4. This paper presents selected parts of the FLUIDE Framework in an example-driven manner by specifying a search and rescue application. It also presents results from an experiment involving 29 potential users exploiting the FLUIDE specifications from the search and rescue application.

THE FLUIDE FRAMEWORK

The FLUIDE Framework is a development environment for professional systems developers containing

- A collection of ready-to-use and highly configurable FLUIDE components supporting flexible composition of end-user solutions for emergency responders
- Composition and configuration approaches
- The FLUIDE specification languages
- A generic mechanism to generate FLUIDE components from FLUIDE specifications
- The FLUIDE method supporting the use of the framework

The FLUIDE Framework offers two specification languages: FLUIDE-A for specifying UIs in an abstract, platform-independent manner, and FLUIDE-D for specifying concrete, platform-specific designs for the FLUIDE-A specifications. FLUIDE-D contains a library of UI patterns that are particularly useful in the emergency response domain, thus meeting R4. FLUIDE-D specifications may automatically be transformed to a running UI. Having specification languages at two different abstraction levels enables combining platform-independent specifications with platform-specific ones in a structured way. Usability of the UIs specified using FLUIDE is ensured through a predictable generation process from compound building blocks and the flexibility due to the component-based approach.

The FLUIDE specification languages meet R1 by providing compound building blocks, which enables common platform-independent specifications across platforms and modalities with large differences. It also allows compact platform-specific specifications of advanced UIs, including UIs where the layout depend on instances at run-time, thus meeting R3. By support of model patterns the compound building blocks are versatile. R2 is met through reflection enabled by embedding domain models in the specifications.

THE SEARCH AND RESCUE CASE

In the case study, we retrospectively specified the UI (denoted *target UI*) of the Generic Ground Station (GGS) application developed as part of the research project DARIUS (http://www.darius-fp7.eu/). In the following, we present examples from the FLUIDE specification. The full description of the search and rescue case including the complete FLUIDE specification is available in (Nilsson and Stølen, 2016).

The GGS application supports emergency personnel managing unmanned vehicles (UVs) as part of an emergency response. The UI of the GGS application (Figure 1) consists of a map display showing a.o. the locations of the incident, the GGS, the UVs, and the search areas. UV search paths and trails are visualized using icons and graphics. Lists of and details for the UVs and missions are shown in separate panes.

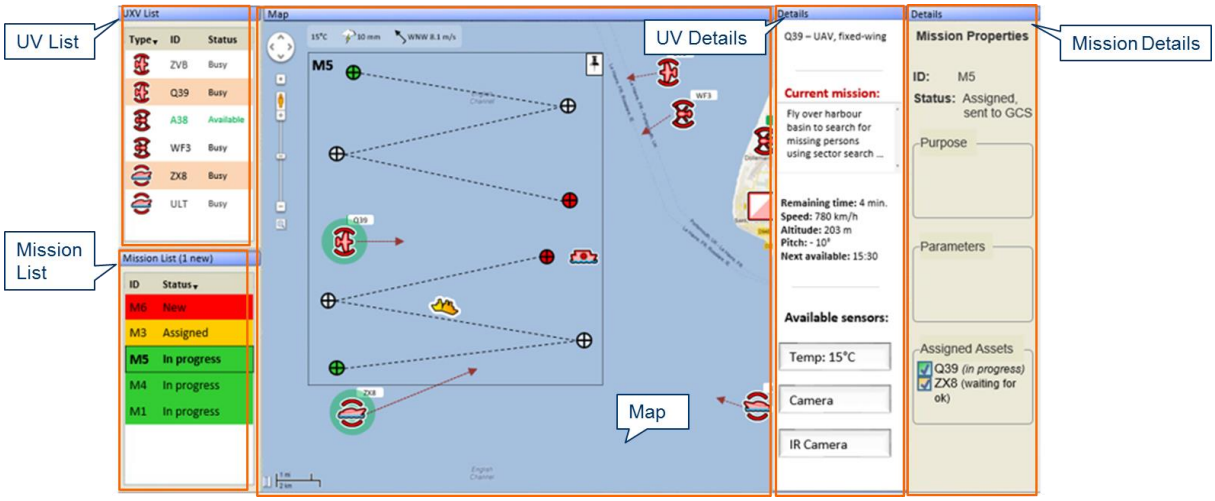

Figure 1. The GGS UI

Mission Presenter in Figure 2 specifies the Mission List and Mission Details panes (both shown in Figure 1) in FLUIDE-A. FLUIDE-A uses a subset of the UML class diagram notation (extended with the anchor symbol identifying the root entity of the model) to express concept models. Annotations specify additional platform-independent visual properties.

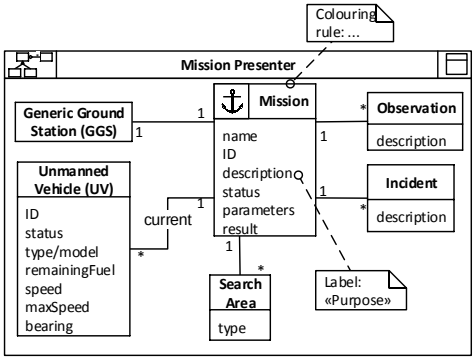

Figure 2. FLUIDE-A specification Mission Presenter

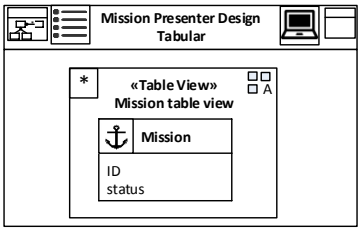

Figure 3. FLUIDE-D specification of the tabular version of Mission Presenter Design

Figure 3 presents a FLUIDE-D design for the Mission List, Figure 4 shows the design for the Mission Details pane, while Figure 5 specifies how UVs are to be displayed as icons on the map (all part of Figure 1). The outer border of a FLUIDE-D specification specifies the UI style(s) and modalities/platform(s) the design targets (PC for GGS). Designs contain one or more *views* in a hierarchical structure. Figure 3, 4 and 5 use Content Views, i.e., views that present instances of one or more entities. Together with Content Integration Views, the available Content Views make up the library of emergency response UI patterns. Such views represent one of the compound building blocks in FLUIDE-D and provide means for specifying advanced designs in a compact way also ensuring usability in the target UI. They provide versatility by being based on model patterns, enabling specification of advanced UIs managing a wide variety of information as long as this information has a structure matching the model patterns of the view. E.g., the *Map Icons View* (Figure 5) provides means for presenting icon-based information in a map UI as long as the model follows a given structure. This view may thus just as well be used for presenting incidents, UVs or victims in the map. Such views combine being specialized and powerful wrt. emergency response needs with being versatile wrt. the actual information they present.

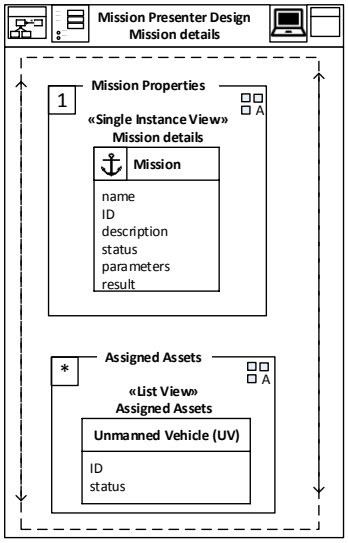

Figure 4. FLUIDE-D specification of the forms-based version of Mission Presenter Design

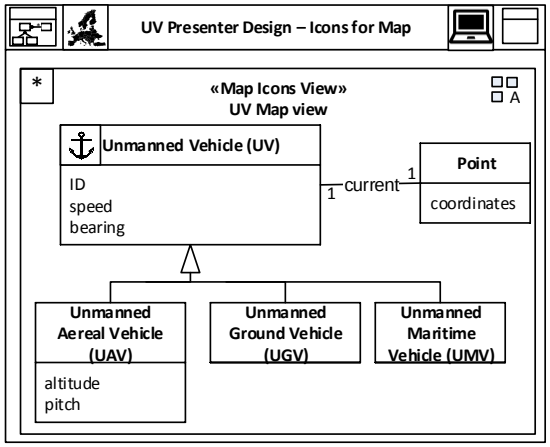

Figure 5. FLUIDE-D specification of map presentation of UVs

EXPERIMENT DESIGN

The overarching research question we addressed in the experiment (Shadish et al., 2002) was: *To what extent are the FLUIDE specifications comprehensible to systems developers?* In particular, we wanted to find out how understandable the specifications are for systems developers with different background and experience, and moreover whether the comprehensibility varies between different constructs in the languages. To address the former, we classified subjects wrt. education level, background and experience. To address the latter, we tested three conditions in the experiment. The subjects included developers, students and researchers. The experiment was conducted in seminar rooms and lasted for one hour. After a short introduction to FLUIDE and the search and rescue case, the participants solved tasks using pen and paper. Then they rated their experience from solving the tasks and provided background information. All questions about knowledge used a Likert scale with 5 values.

There was in total 29 participants, 20 males and 9 females. The age varied from 20 to 58, average 34 (sd=9,4). Table 1 shows the participants' knowledge profile. 13 had formal background or experience within model-driven development (MDD), 13 did not (yes/no).

	No knowledge	Minor knowledge	Some knowledge	Good knowledge	Expert
Knowledge of UML class models	1	11	4	11	2
Knowledge of UI design and usability	3	8	9	6	3
Knowledge of emergency response work	14	7	2	4	1

Table 1. Knowledge profiles

The following three conditions were investigated in the experiment:

(i) both FLUIDE-A and FLUIDE-D specifications were shown

(ii) only FLUIDE-D specifications were shown

(iii) only FLUIDE-A specifications were shown

There were 36 tasks in the experiment (12 for each condition). The subjects were asked to match FLUIDE specifications with parts of the target UIs using different set-ups for each condition. The participants were also given a one-page explanation of the most important constructs in the FLUIDE specification languages.

Questions about difficulty were answered using a Likert scale with six values. Questions about the integration of UML class diagrams in the specifications were answered using a Likert scale with five values. The student-participants received a small gift as gratitude, and some prices were drawn among the participants.

To operationalize the research question, we formulated some hypotheses about the outcome of the experiment:

- H1: Competence in using modelling languages will improve the score in the test
- H2: The participants' assessment of difficulty of the tasks will correlate with the score in the test
- H3: Scores for the different conditions will vary, more precisely:
    - H3.1: Score for condition (i) will be higher than for condition (ii)
    - H3.2: Score for condition (ii) will be higher than for condition (iii)
    - H3.3: Score for condition (i) will be higher than for condition (iii)

The full description of the experiment design, as well as more a more detailed presentation of the findings from it are available in (Nilsson and Stølen, 2016).

FINDINGS FROM THE EXPERIMENT

Task Scores

For each task, the participants' answers were given a score of 0 if wrong and 1 if correct. All scores reported in the following are averages. The average of all participants' (average) score is 0,57 (sd=0,13). The highest score obtained was 0,81, the lowest was 0,21. The correct answer was the chosen by most participants for 29 of the 36 tasks. For 22 of these tasks, a majority of the participants answered correctly. The most difficult task had an average score of 0,19 (sd=0,40), while the easiest task was answered correctly by all.

Assessment Questions

The answers to the questions about difficulty are summarized in Table 2.

	Impossible	Very difficult	Somewhat difficult	Quite easy	Very easy	Trivial
Were the specifications easy to understand?	0	7	15	5	1	0
Was it easy to grasp the essential part of the specifications?	0	2	10	14	2	0
Was it easy to understand the connections between the specifications and the UIs?	0	4	13	10	1	0

Table 2. Assessment of difficulty

Most participants did not react to the mixing of UML class diagrams and FLUIDE-specific notation, and among the participants who reacted, a majority reacted positive.

Findings Related to the Hypotheses

We assessed H1 by investigating the scores in relation to UML knowledge, as UML class diagrams are used in FLUIDE. The results from this shows a weak, but not significant trend that scores increase with UML knowledge. We found though that the participants with background or experience in MDD had a significantly higher score (0,63, sd=0,11) than the ones without this background (0,52, sd=0,14). An equal variance t test shows $t=2,461$ (24) $p<0,05$.

When investigating H2, we used the average assessment for the three ratings of difficulty (Table 2). The three ratings included in the measure had acceptable inter-item reliability (Cronbach's alpha 0,75) for doing this. When comparing the difficulty rating for the participants with background or experience in MDD (cf. H1) with the rest, we saw that the MDD group found the specifications significantly easier to understand. An equal

variance t test shows t=2,215 (24) p<0,05.

To investigate H3.1, we compared the average scores for the tasks within condition (i) (0,58, sd=0,18) with the similar scores for the tasks within condition (ii) (0,63, sd=0,16), i.e., the score for condition (i) was lower than for condition (ii). The t test shows that the difference is not significant.

To investigate H3.2, we compared the average scores for the tasks within condition (ii) (0,63, sd=0,16) with the similar scores for the tasks within condition (iii) (0,51, sd=0,19). A paired t test shows that this difference is significant: t=3,452 (28) p<0,01.

To investigate H3.3, we compared the average scores for the tasks within condition (i) (0,58, sd=0,18) with the similar scores for the tasks within condition (iii) (0,51, sd=0,19). A paired t test shows that this difference is significant: t=1,905 (28) p<0,05.

DISCUSSION

As the introduction to the FLUIDE languages lasted in total approximately 5 minutes, and solving the tasks was performed under time pressure, we argue that the experiment was conducted under conditions where we could expect rather low scores. Under this assumption, we rate the score level and the share of tasks with most correct or a majority of correct answers as an indication that understanding FLUIDE specifications is highly achievable for systems developers.

The difficulty scores show that it is possible to match FLUIDE specifications with UIs without fully understanding the specifications. This indicates that appropriate training will ensure good understanding. The response to combining UML class diagrams and FLUIDE-specific notation provides support for an important design choice.

It is relatively simple for participants without much UML competence to match elements in the UML models and UI annotations with labels and headings in the UIs. This type of *lexical matching* may be one reason why UML knowledge is not sufficient to explain differences in understanding, as this raises the scores for the participants with low UML knowledge. Because the MDD group is familiar with seeing connections between abstract models and concrete computer systems they are able to match also without lexical support.

FLUIDE-A specifications often contain model elements not shown in the target UI, making lexical matching less effective and sometimes even misleading. FLUIDE-D specifications provide no such additional information. This may explain why H3.1 is not supported, i.e., the reason why including the FLUIDE-A specifications has no effect on comprehensibility is that the larger and more complex models confuse systems developers. The results from the experiment do not support H3.1, and we conclude that to obtain an understanding of the match between a FLUIDE specification and the corresponding UI, it is sufficient to use FLUIDE-D specifications. The main rationale behind formulating H3.1 was that the annotations, which are part of FLUIDE-A but not shown in the FLUIDE-D, would increase understanding. We have no indications that the presence of UI annotations has a negative effect. Thus, a natural enhancement of the FLUIDE-D specifications is to add UI annotations.

The conclusions in this paper regarding comprehensibility may be generalized to other languages distinguishing between platform-independent and platform-specific specifications. I.e., the platform-specific specifications are easier to understand, and to further enhance the understanding they should be self-contained.

RELATED WORK

As argued in the introduction, the most influential languages and approaches supporting model-based UI development do not fully meet all the requirement we have identified. The same holds for OMG's IFML standard. In particular, none of the assessed approaches meet the requirement of having compound building blocks, and only domains related to emergency response are supported.

Research on ICT solutions for emergency response focuses mainly on developing concrete systems, usually also addressing their UI. There is also focus on modelling different aspects of emergency response, like tasks, procedures, organization and standards – often operationalized through ontologies. Furthermore, there is also some focus on development methods, but surprisingly little on frameworks and tools for developing emergency response applications. A noteworthy exception is the framework presented in (Fitrianie and Rothkrantz, 2009). This framework supports development of multimodal UIs. Even though UML class diagrams play a role in the framework, specifications are not expressed at an abstract level. The ISyCri project (Truptil et al., 2009) makes use

of the distinction between platform-independent and platform-specific models in their ontology-based approach. Their focus is not on development, though, but rather on facilitating collaboration. Balasubramanian et al.'s (2005) domain-specific language for specifying emergency response systems does not support UI development.

CONCLUSIONS AND FUTURE RESEARCH

In this paper we have presented the FLUIDE Framework, focusing on the FLUIDE specification languages. These languages offer a unique combination of feature making them suited for specifying traditional emergency response applications as well as ready-to-use and highly configurable components. Such components support flexible composition of UIs for emergency responders.

We have used FLUIDE to specify the UI of an application supporting management of unmanned vehicles in search and rescue operations. We have also conducted an experiment involving 29 systems developers with varying background to assess the comprehensibility of these specifications for systems developers not knowing FLUIDE. The main conclusions are:

- FLUIDE is well suited for specifying UIs in the search and rescue case
- Understanding FLUIDE specifications is highly achievable for systems developers
- It is possible to match FLUIDE specifications with UIs without fully understanding the specifications
- Our approach of using UML and task modelling notation as part of the FLUIDE notation enhances the understanding
- Competence in model-driven systems development increases understanding significantly
- To obtain an understanding of the match between a FLUIDE specification and the corresponding UI, it is sufficient to use FLUIDE-D specifications
- UI annotations from the FLUIDE-A specifications should be included in the FLUIDE-D specifications
- To ensure understanding of specifications from languages distinguishing between platform-independent and platform-specific specifications, self-contained platform-specific specifications should be used

Among our planned future research is to complement the framework, including tool support and adaptation mechanisms, and conducting experiments where systems developers use the FLUIDE Framework to specify UIs.

ACKNOWLEDGMENTS

The work on which this paper is based is supported by the EMERGENCY project (187799/S10) funded by the Norwegian Research Council and the following project partners: Locus AS, The Directorate for Civil Protection and Emergency Planning, Geodata AS, Norwegian Red Cross, and Oslo Police District.

REFERENCES

1. Balasubramanian, K. et al. (2005). A platform-independent component modeling language for distributed real-time and embedded systems. *Proc. 11th Real Time and Embedded Technology and Applications Symposium.* IEEE.
2. Fitrianie, S. and Rothkrantz, L. (2009). Computed Ontology-based Situation Awareness of Multi-User Observations. *Proc. ISCRAM'09.* ISBN 978-91-633-4604-0.
3. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. and López-Jaquero, V. (2004). USIXML: a language supporting multi-path development of user interfaces. *Proc. of EHCI-DSVIS'04.* Springer.
4. Navarre, D. et al. (2009). ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Trans. Computer-Human Interaction 16(4).*
5. Nilsson, E.G. and Stølen, K. (2010). Ad Hoc Networks and Mobile Devices in Emergency Response – a Perfect Match? *Proc. 2nd International Conference on Ad Hoc Networks*. Springer.
6. Nilsson, E.G. and Stølen, K. (2011). Generic functionality in user interfaces for emergency response. *Proc. OZCHI'11*. ACM.

7. Nilsson, E.G. and Stølen, K. (2016). The FLUIDE Framework for Specifying Emergency Response User Interfaces Employed to a Search and Rescue Case. *SINTEF Report A27575*. ISBN 978-82-14-05931-1.
8. Paternò, F., Santoro, C. and Spano, L.D. (2009). MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Computer-Human Interaction 16(4).*
9. Shadish, W.R., Cook, T.D. and Campbell, D.T. (2002). Experimental and Quasi-experimental Designs for Generalized Causal Inference. *Houghton Mifflin.*
10. Truptil, S., Benaben, F. and Pingaud, H. (2009). Collaborative process design for Mediation Information System Engineering. *Proc. ISCRAM'09.* ISBN 978-91-633-4604-0.