An Aspect-Oriented Tool Framework for Developing Process-Sensitive Embedded User Assistance Systems

Bedir Tekinerdoğan[1], Serap Bozbey[2], Yavuz Mester[1], Erdem Turançiftci[1], and Levent Alkışlar[2]

[1] Bilkent University, Department of Computer Engineering, 06800 Bilkent Ankara, Turkey
{bedir,mester,erdem}@cs.bilkent.edu.tr
[2] Aselsan, PO. Box. 1, 06172, Yenimahalle, Ankara, Turkey
{bozbey,alkislar}@aselsan.com.tr

Abstract. Process-sensitive embedded user assistance aims to provide the end-user the necessary guidance based on the state of the process that is being followed. Unfortunately, the development of these systems is not trivial and has to meet several challenges. The main difficulties appear to be related to integration of process-sensitive guidance in the application and the crosscutting behavior of help concerns. To address these issues we developed an aspect-oriented tool framework *Assistant-Pro* that can be used to develop process-sensitive embedded user assistance for multiple applications. The framework provides tools for defining the process model, defining guidance related to process steps, and modularizing and weaving help concerns in the target application for which user guidance needs to be provided. The framework has been developed and validated in the context of Aselsan, a large Turkish defense electronics company.

Keywords: context-sensitive user assistance, aspect-oriented software development, industrial applications.

1 Introduction

User assistance is a broad term which refers to the guided assistance to a user of a software product, to help accomplishing tasks and ensure a successful user experience [12][23][16]. The traditional form of user assistance is an off-line printed user manual that is separate from the system. User assistance has now evolved to *online help* systems [30][31][35] that provide information to the user in an electronic format and which can be opened directly in the application. Until recently, online help systems usually have adopted a topic-oriented approach in which help can be requested based on keywords. The corresponding assistance is presented in different formats such as html, text, or PDF.

Embedded user assistance systems can be defined as a special form of online help in which the documentation of the application resides within the application [26]. The key motivation for embedded user assistance is the fact that traditional, separate user

S. Katz et al. (Eds.): Transactions on AOSD VIII, LNCS 6580, pp. 196–220, 2011.

assistance is inherently reactive. This means that users only consult the documentation when they do not know how to proceed. The result is that they stop what they are doing, open the documentation, find the information they are looking for and then return to the application. Research on user assistance has shown that it is basically because of this separate effort and disruption of the user's flow of work, that users are reluctant to using help [4][9].

An important category of embedded user assistance systems are *context-sensitive user assistance* systems [5][26][36]. In context-sensitive user assistance, help is obtained from a specific point in the state of the software, providing help for the situation that is associated with that state. In contrast to general online user assistance, context-sensitive assistance does not need to be accessible for reading as a whole. In general the system is defined as a set of states to which a topic is related that extensively describes the corresponding state, situation, or feature of the software. Context-sensitive help can be provided in different ways including automatic tooltips over controls, notifications in the status bar, or by new panes which are opened after explicitly clicking a button. Context-sensitive help systems, such as Microsoft's WinHelp and Sun's JavaHelp [23] have been applied to various kind of software systems. The advantage of embedded, contextual help is that it can provide immediate assistance to users without having to leave the context in which they are working. This is important because, as noted above, users seem to be very often reluctant to use help that is not integrated [4][9].

One category of context-sensitive user assistance systems focuses on defining help based on the process state, and we define these as *process-sensitive user assistance systems*. In this paper, 'process' implies the steps that need to be followed while using a particular application. The offered help content depends on the steps that have been processed. An example of such applications is a safety-critical system in which a strict process needs to be followed in order to avoid faulty behavior.

Unfortunately, developing embedded context-sensitive user assistance systems in general and process-sensitive systems in particular, is not trivial and has to meet several challenges. First of all, it appears that user-assistance concerns usually cannot be easily localized in single modules and as such tend to crosscut multiple modules. A common problem of crosscutting concerns is that they reduce the modularity of the system and as such impede maintenance. User assistance design can benefit from aspect-oriented software development (AOSD) approaches [8] to modularize the crosscutting concern and support maintenance. Another important problem is the reuse of user assistance tools for different applications. In general, the need for modularizing crosscutting concerns is motivated in case of change of crosscutting concerns for the *same* application. However, in this paper we will also show that this might be required for multiple applications at the same time ('change in time vs. change in space'). Rather than developing custom-based help assistance for each separate application, it is required to support reuse of a help concern and as such reduce the time of development.

To tackle these problems we developed an application framework and the related tools for modularly extending applications with context-sensitive help. The framework has been developed within the industrial context of Aselsan [1], Turkey, which

is a leading high technology, multi-product defense electronics company introducing state-of-the-art equipment and software intensive systems solutions for both sophisticated military and professional applications.

The remainder of the paper is organized as follows: In Section 2 we describe the problem statement in more detail using the industrial cases of Aselsan. In Section 3, we define the conceptual model that represent and integrates concepts of process modeling and guidance modeling. In Section 4, we present the architecture of the tool and the required flow of control for using the tools. Section 5 discusses the implementation of the guidance aspect and the weaving process. In Section 6, we present the evaluation of the framework and finally in Section 7 we conclude the paper.

2 Case Description and Problem Statement

Since context-sensitive user assistance add-ons enhance the support for guidance of systems, they have gained importance in recent years. Unfortunately, the development of such context-sensitive user assistance systems also introduces new challenges that have not yet been explicitly considered in the user assistance domain. As stated before, our main target is the *development* of embedded user assistance systems that provide guidance with respect to the state of the process. To clarify the problems, in the following Section 2.1 we will first describe two pilot project applications that have been developed at Aselsan [1], and which also require guidance to the flow of actions. In Section 2.2 we explain the approach that was used at Aselsan to define user assistance. In Section 2.3 we will list the problems of context-sensitive embedded user assistance systems using the two applications.

2.1 Case Description

Example Case Application 1 – Message Management System
Message Management System (MMS) is a kind of e-mail application that helps two or more connected peers to asynchronously send and receive messages among each other and manage messages inside its own mailbox. A screenshot of the application is shown in Fig. 1. By using this application, each peer can view its incoming messages and can reply, forward, or delete them. Also the peer can create a new message, send it to the other peer and if required save this as a template. Messages are listed in the user interface according to their status. All the information about messages are stored in a database.

Example Case Application 2 - Listing and Listening of Voice Records
The *Listing and Listening of Voice Records Application (LLVR)* is used to organize and play the voice records which are stored in different audio formats. By using this application, users can query stored voice records from the database, review all the results as a list or get the print of the list. The selected records can be played, paused, stopped, or replayed from the list. By defining the start and finish times, any part of a record can be replayed. The operator can also attach a text note about the

record to the database. The records can be imported from and exported to a database. Again, this application has no guidance to users. The user needs explicit guidance to operate the application effectively and to reduce the potential faults due to misuse.

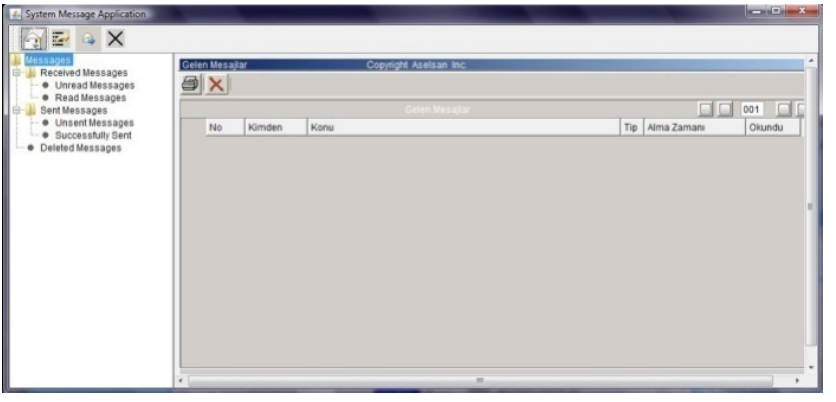

Fig. 1. Screenshot of Message Management System (MMS)

2.2 Current Approach

The above mentioned goals of user assistance and the need for embedded user assistance have also been encountered for the commercial applications that are developed by Aselsan [1]. In many of these applications, it is required that the end user follows a certain sequence of steps in order to complete the corresponding scenario successfully. For defining help, Aselsan adopted both external manuals and electronic help in its products depending on the nature of the application.

To define user assistance for the applications in Aselsan, the earlier conventional approach includes three steps. First, a senior system engineering team develops a scenario in a textual format to fulfill the requirements of the user to achieve a certain task. The textual format includes the control flow of the scenario including alternative flows and unexpected terminations or transitions to other scenarios. In the second step the development team implements the scenarios in the code and UI including the selection of graphical elements and the implementation of the transitions that are defined in the scenarios. In the third step, user assistance specialists write the user manuals for the application in both electronic and hardcopy format. A link is defined in the application to the electronic user manuals. The format of this help is basically defined as PDF documents or wiki pages. It is implicitly required that the users know the scenario as defined in these documents when operating the system. In case the user requires help they need to open the PDF documents and search for the information that solves their problem. To support the definition of help also third party tools, such as *Fast-Help* [11] is used. *Fast-Help* is a Windows Help File Generator that produces online and offline documentation in electronic formats such as HTML, PDF or .HLP. Although these third party products help to better organize the help files and provide better presentation and query mechanisms, the provided help is still not

embedded in the application. As such, the earlier mentioned problems of external help still remain. Due to this problem, users very often demand further training to operate the system effectively. This also brings additional undesired costs to the company.

2.3 Problem Statement

Several researchers have focused on the challenges of embedded user assistance systems. In general these seem to have focused basically on the perspective of the user assistance specialist whose primary responsibility is to design and improve the layout of the system [34][12]. This has resulted in different guidelines and design checklists for improving the usability of embedded user assistance systems. In the following we list the obstacles that are related to the *development* of context-sensitive user assistance systems.

1. Crosscutting Help Concern

One of the reasons for the difficulty of developing and maintaining embedded user assistance is the crosscutting nature of help concerns. User assistance might be required for different places in the code. Very often the need for user assistance is triggered by some widget event, such as clicking a button, selecting a checkbox or radio button. A close analysis of the applications shows that the widget events and as such the need for user assistance places, is not localized but is scattered over multiple modules. Consider some example code snippets in the application as given in Fig. 2.

```
...
ApplicationFrame appFrame = new ApplicationFrame();
...
public boolean replyMessageButtonClicked() {…}
...
new SaveTemplateWindow();
public boolean sendButtonClicked() {…}
public void windowClosing(WindowEvent arg0) { …}
....
public void closeButtonClicked() {…}
....
```

Fig. 2. Example of the code snippets of the MMS application indicating events that need to be monitored and for which help needs to be provided

To provide process-sensitive help, we need to track all such events in the code that can lead to a state behavior. In fact these events can be anywhere in the code and often cannot be put in a single module. Moreover, since both the application code and the provided help might need to change regularly, the maintenance of the code and the provided help might become cumbersome and too costly.

The problems of crosscutting concerns have been widely addressed in the aspect-oriented software development (AOSD) community [8]. AOSD provides explicit abstractions to localize and model crosscutting concerns and as such support maintenance and reuse of these concerns. In this context it is beneficial to implement help concerns as aspects and provide a way to compose these with the application tools.

2. Reuse across multiple application tools

Different applications are used in Aselsan that require embedded user assistance based on process context. Typically, we can identify the following categories of applications (1) legacy applications with help implemented, (2) legacy applications without help implemented, and (3) new applications that require integrated help. In general, while implementing help concerns, the same kind of constructs need to be implemented for the different types of applications. For example for both the small applications MMS and LLVR that have been explained in the previous section we need to provide guidance which is typically implemented as status bar descriptions and textual explanations in separate windows. To optimize the effort for implementing help concerns across multiple applications, it is necessary to adopt the right modularization and reuse approaches.

Also in this case AOSD seems to be invaluable to model the crosscutting concerns not only for a single application but across applications. Typically this will require defining an abstract guidance aspect for monitoring the process and providing appropriate help. The abstract aspect might need to be extended for the different applications to provide the required help.

3. Implicit Process

Although embedded user assistance can provide the user with help at the right moment, the related process is still implicit. In general, the required process is not explicitly modeled but is implicitly defined by the applications. This complicates tracking and monitoring the user when realizing the tasks. We can observe this problem in both the MMS and LLVR applications. Although a specific process needs to be followed to complete the task, the process is not defined within the application but resides in external documents.

To improve the understandability, reuse, and management of the adopted process we need to explicitly model the process steps that need to be followed in order to complete the task successfully

4. Tool Support and Presentation Issues

One of the key problems in defining embedded user assistance systems is the lack of tools and standard techniques. To the best of our knowledge there is no framework or tool available to support the development of embedded user assistance systems in a modular way. These tools need to support the implementation of user assistance but also take care of the space problem while displaying help in the application window. Traditional user assistance systems which provide off-line documentation do not have space restrictions. However, since help needs to be integrated in embedded user assistance systems, the space for displaying the help becomes inevitably a problem [34]. In general, user assistance specialists are involved in the design of the user interface from the very start to properly design the layout of the user assistance, and as such avoid having to retrofit assistance to an interface later on. Unfortunately, this does not seem to be sufficient and too static to cope with the requirements for changing user assistance later.

To provide systematic support for integration of different help concerns across applications, we aim to provide a framework-based approach. A framework is a reusable, "semi-complete" application that can be specialized to produce custom applications [10]. In our case, it will need to include the abstractions of context-sensitive embedded user assistance which will include modeling the process and the required help.

3 Process-Sensitive User Assistance Model

To provide context-sensitive help based on process state, it is necessary that we model the process first. Process modeling has been addressed in different domains including method engineering and meta-modeling [33]. One of the important process modeling approaches is the OMG Software Process Engineering Meta model (SPEM) [32]. This meta-model is used to describe a concrete software development process or a family of related software development processes. The SPEM specification is deliberately limited for defining the minimal set of process modeling elements necessary to describe any software development process, without adding specific models or constraints for any specific area or discipline. As such it does not include explicit and

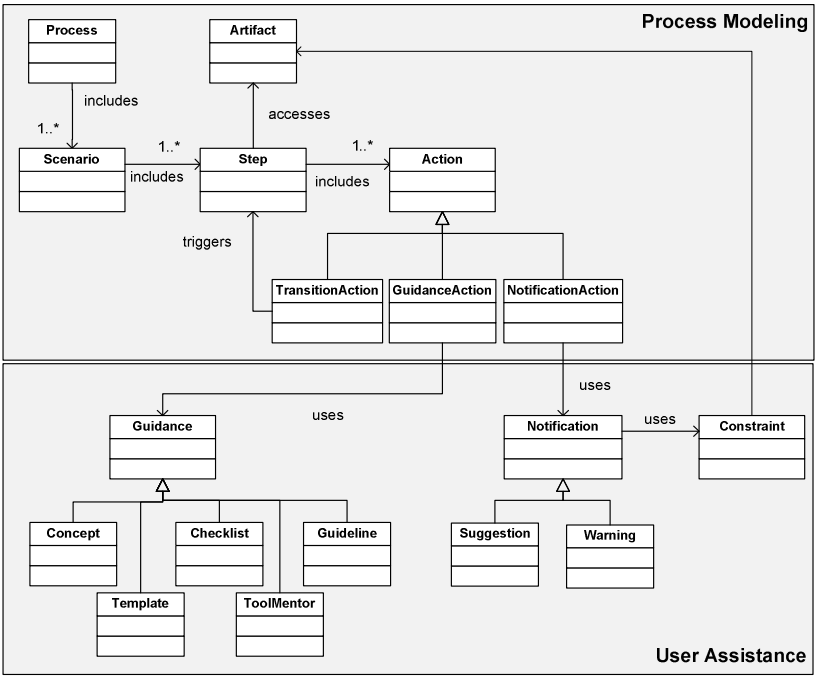

Fig. 3. Process-Sensitive Embedded User Assistance Model

elaborate notions for user assistance concepts. Inspired from SPEM, we have defined a model for process-sensitive user assistance. The model as depicted in Fig. 3 consists of two parts *Process Modeling* and *User Assistance*.

In the *Process Modeling* part, the concept *Process* represents the ordered set of scenarios to accomplish a particular goal in the application. *Scenario* is a particular sequence of steps taken to complete a process. A process may be completed following any of a number of scenarios associated with it. *Step* includes *Actions* that can be triggered through user interface components. There are three types of actions. *Guidance Action* triggers guidance for the current step. *Transition Action* facilitates transition between steps. *Notification Action* defines the notification.

The part *User Assistance* of the model in Fig. 3 defines the concepts necessary for user assistance modeling that can be provided either through *Guidance* or by giving *Notification. Guidance* represents the type of guidance that is provided to the user and defines the types *Concept*, *Checklist*, *Template, ToolMentor* and *Guideline. Concept* defines the textual explanation about the step. *Template* defines a predefined document that provides a standardized format for a particular process. *Checklist* is a list of steps that need to be performed to complete a particular scenario. *ToolMentor* shows how to use a specific tool to accomplish a process. *Guideline* is a set of rules and recommendations. *Notification* defines the notification to the user about the validity of the artifacts and can be either *Suggestion* or *Warning. Constraints* are checked when subjected to artifacts to determine if a notification is needed.

4 Tool Architecture and Adopted Approach

We have developed the tool framework *Assistant-Pro* that aims to support the development of process-sensitive embedded user assistance. The tool architecture and the workflow of *Assistant-Pro* are given in Fig. 4. The flow of control is indicated through numbered circles in the figure. The figure shows both, the part of the *Framework* and *Application*. The framework includes the tools, *Process Definition Tool* and *Help Definition Tool* and largely builds on the model as defined in Fig. 3. The tool framework supports four key stakeholders including *Code Analyzer, Process Definer, User Assistance Specialist* and *End-User. Code Analyzer* analyzes the code of the application and annotates this to support the aspect that is generated later on. The *Process Definer* uses the tool *Process Definition Tool* to model the process that need to be followed. Subsequently, the *User Assistance Specialist* uses the *Help Definition* Tool to define the help for the corresponding process steps. The *End-User* will use the application that is extended with help. In the following subsections, we describe the important processes in detail. In Section 4.1, we first explain the process of annotating the application code. In Section 4.2, we explain the modeling of the process using the *Process Definition Tool*. In Section 4.3, we describe the description and association of help to individual steps in the application.

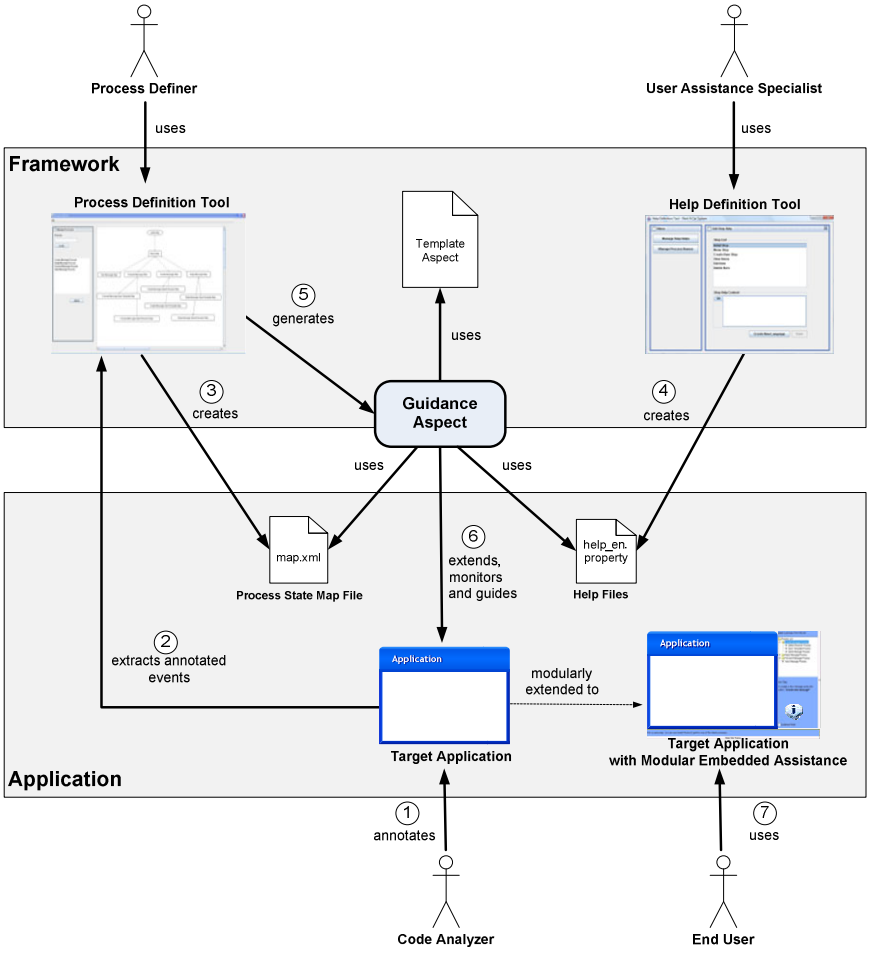

Fig. 4. Tool Architecture and workflow of *Assistant-Pro*

4.1 Analyzing the Code

As depicted in Fig. 4, the first step in the process is the analysis of the application code. Initially, the *Code Analyzer* will annotate the constructor of the main UI component in the application with the annotation *@ApplicationInitialization*. This will be needed later on to initialize the necessary help module and associate a reference of the code with the guidance aspect. Fig. 5 shows an example of the annotated code for the MMS case. Line 2 annotates the method call *mainframe()* with the annotation *@ApplicationInitialization.*

Further, the *Code Analyzer* will analyze the code and determine the different user interface states of the application that correspond to relevant steps in the process to be followed. In parallel, the *Code Analyzer* will determine the method calls (events) in the code that lead to different state transitions. The *Code Analyzer* will annotate these

methods with the predefined *@Event()* annotation which has the property *name* to define a unique and relevant name. In Fig. 5 the method call viewMessageButtonClicked() is annotated with *@Event* and the name "View Message Button Clicked" is given. Note that all the annotation names that we have used here will also appear when defining the process state map in the *Process Definition Tool* in Fig. 6.

```
1.  ....
2.  @ApplicationInitialization
3.  public mainFrame() {
4.  ....
5.  .....
6.  @Event (name = "View Message Button Clicked")
7.  public void viewMessageButtonClicked() {  …… }
```

Fig. 5. Example of Annotated Application Code

4.2 Modeling the Process Using Process Definition Tool

The *Process Definition Tool* is used by the *Process Definer* to define the required process that needs to be adopted in the application for which help needs to be provided. A snapshot of the tool together with the related steps is depicted in Fig. 6. In the given example the process definition for MMS has been given. To define the process the following actions need to be performed with the tool: (1) extract annotation names from application code, (2) define the state abstractions of the user interface, (3) define the transitions between states, (4) associate transitions with the annotated events of the application, (5) define the processes that represent a set of scenarios.

Extracting annotations from the application is triggered by entering the location of the application when the project is created for the first time (step ② in Fig. 4). *Process Definition Tool* parses the corresponding file, looks for annotated events and describes these in a list that is presented to the *Process Definer.*

Process Definer will define the process by defining the states and the transitions among these states (step ③ in Fig. 4). An example state is *Create Message State* which defines the UI state in which the end-user needs to create a message. The states for the MMS application are shown in the right pane of the tool. The rectangles in the figure represent the states to accomplish the process; the arrows represent the transition between these states. States and transitions can be generated by right clicking the mouse button. Entering a state defines the name of the state. Entering a transition defines the transition between two states, and also associates the transition with the events in the application code. Transitions are associated to the events by referring explicitly to annotations that were derived from the application. By using annotations, a loose coupling with the application code is realized and likewise the *Process Definer* does not need to be bothered about the exact signature of the events in the code that might trigger a state change.

The output of the *Process Definer* is the *Process State Map* file (map.xml in Fig. 4) which defines the total set of states and the transitions. In the given example the process state map starts with *Initial State*, a dummy state to indicate that the application has not started yet. After the MMS application is started, *Menu State* will follow, and from here the user can either go to the *View Message State, Forward Message*

State, Create Message State, and *Reply Message State. Forward Message State* can be followed with *Save Template State, and Select Receiver State*. *Create Message State* and *Reply Message State* can follow with *Select Receiver State* and *Save Template State*.

In essence, the process state map defines thus the potential number of paths (of state transitions) that the application can traverse. However, not all paths will need to be followed in the end-user application. In the tool we can describe the specific processes that the user has to follow. Each process represents a set of scenarios, thus a specific path traverses through a number of states in the process state map.

An example of such a process as defined in Fig. 6 is *Create Message Process.* For each process we define the name and the final state. For *Create Message Process* typically the final state is defined as *Create Message Save Template State.* The user can be in any state of the program and as such there are usually multiple ways, scenarios, to get to the end state *Create Message Save Template*. While the user will traverse through the states to achieve the final goal, the tool will provide guidance and help as defined in the next section.

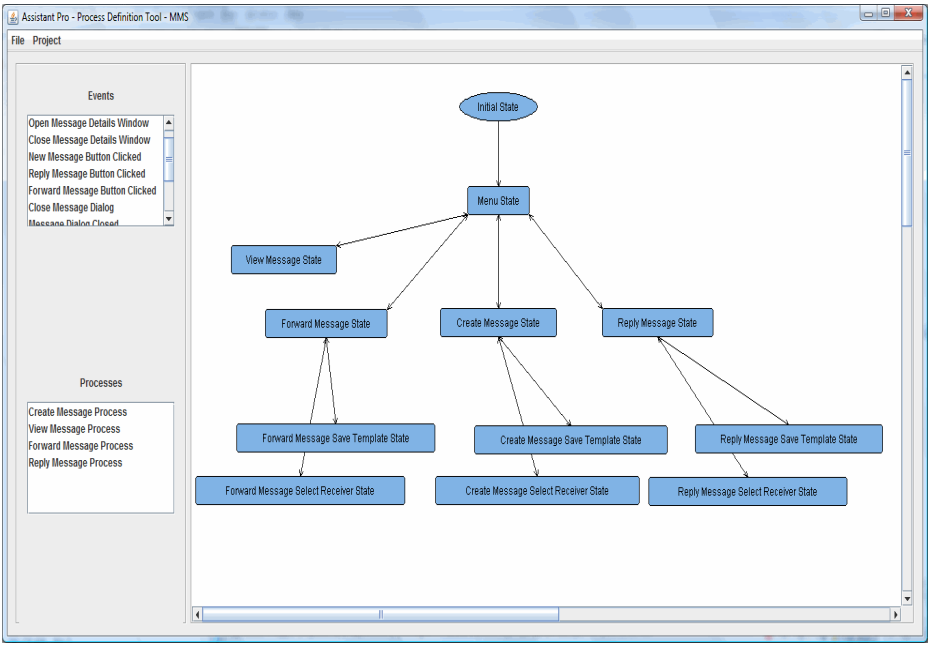

Fig. 6. Snapshot of the Process Definition Tool (for MMS)

4.3 Defining Help Using Help Definition Tool

The role of the *Help Definition Tool* is to define user assistance elements for the states and events (step ④ in Fig. 4). A snapshot of the tool is shown in Fig. 7. In the tool three actions can be performed: *Edit State Hints, Edit Process Names, Edit Event Advices.* By clicking *Edit State Hints,* the textual help description related to a given

step can be defined. In Fig. 7, for instance, *Menu State* is selected, and the help description is given in the text field below the menu. *Edit Event Advices* defines the description of the action that the user should do next for each event. *Edit Process Names* is used to describe the process that should be presented to the user. In this way, the user will be provided an explicit description of the process and can follow his/her actions.

The tool also supports the definition of help for multiple languages that are listed in a list box. The *User Assistance Specialist* can create another language and also remove a language by using the tool. For each of the languages in the list the *Help Definition Tool* creates a properties file and saves the *Help Definition.* The help for each Step for each supported language is listed in a tabbed content pane so that it can be viewed, edited, or removed. The *User Assistance Specialist* can select the state from the list, select the language from the supported languages, and enter the help descriptions.

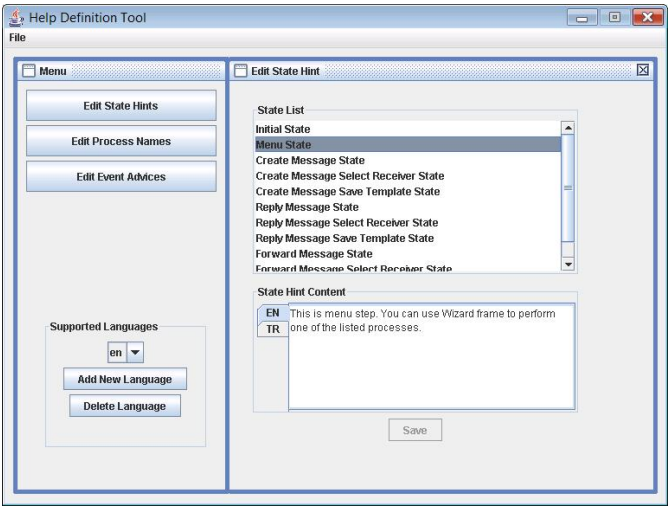

Fig. 7. Snapshot of the Help Definition Tool (for MMS)

5 Implementation and End-User Experience

In this section, we will elaborate on the implementation of the Guidance Aspect in Section 5.1, and in Section 5.2 explain the end-user experience after weaving the aspect.

5.1 Implementation of Guidance Aspect

The guidance of the end-user application is defined by the aspect *Guidance Aspect* as depicted in Fig. 4. In the figure the aspect is located on the border of the framework

part with a link to the application in the application part. This is because the aspect is generated (step ⑤ in Fig. 4) within the tool framework, while it will monitor the application at run-time (step ⑥ in Fig. 4).

We have defined an abstract *Guidance Aspect* as given in Fig. 8. The aspect provides abstract methods for reading the process state map file and help file (Line 14-15). Further the aspect defines the abstract pointcuts `mainWindow-Initialized()` (line 7) and `eventTriggered(Event event)`(line 17). The pointcut `mainWindow-Initialized()` is used for capturing the initialization of the application and for initializing the GUI components. The initialization is done using an abstract method `initializeMainWindow(mainWindow)`. The pointcut `eventTriggered(Event event)` aims to capture the widget events in the application that can cause a transition. It first checks whether the event has resulted in a state change, and if this is the case it will update the help (line 22). The method `updateModule()` will access the help files and retrieve the help that will need to be presented to the user.

```
1.  package helpmodule;
2.  import java.awt.Component;
3.  import helpmodule.processstatemap.ProcessStateMap;
4.
5.  public abstract aspect GuidanceAspect {
6.  ProcessStateMap map;
7.  abstract pointcut mainWindowInitialized();
8.
9.  after() returning (Window mainWindow): mainWindowInitialized()
10. {  initializeMapInstance();
11.    map.updateState(map.getInitialEvent());
12.    initializeModule(mainWindow);}
13.
14. abstract void initializeMapInstance();
15. abstract void initializeModule(Window mainWindow);
16.
17. abstract pointcut eventTriggered(Event event);
18. Object around(Event event) : eventTriggered(event)
19. {  Object eventOutput = proceed(event);
20.       boolean stateChanged = map.checkStateChange(event.name(), eventOutput);
21.           if (stateChanged)
22.           updateModule();
23.           return eventOutput; }
24. abstract void updateModule(); }
```

Fig. 8. Abstract Guidance Aspect

The abstract aspect can be manually specialized by defining a concrete aspect in which the pointcuts, the advices and the necessary code can be implemented. For example, in Fig. 9, *Concrete Manually Defined Guidance Aspect-LLVR* has been defined for the LLVR system as explained in Section 2.1. However, since the points at which help will be provided have been defined earlier on using the *Process Definition Tool* and *Help Definition Tool*, we could implement a generator program that can automatically generate the concrete aspect from the given input. The mechanism for generating the aspect is shown in Fig. 9.

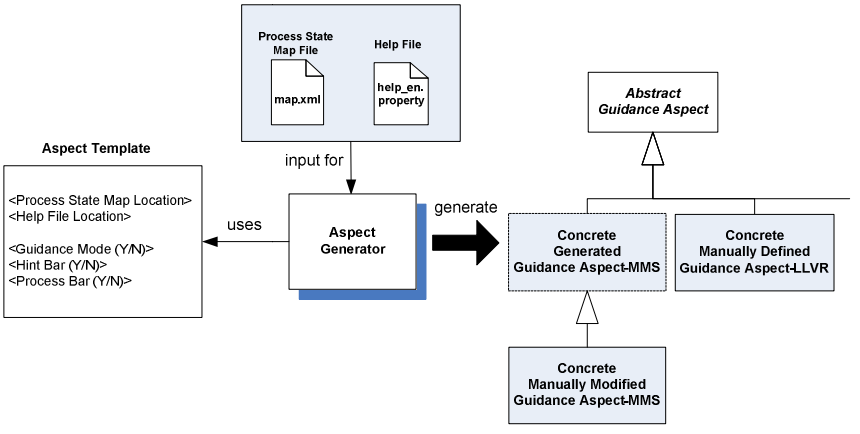

Fig. 9. Generation of Concrete Aspect approach

The input for the generator represents the *Process State Map File* (map.xml) and *Help File* (help_en.property) which have been created before using the *Process Definition Tool* and *Help Definition Tool* as defined in Fig. 4. Further, the generator uses a predefined template in which the aspect codes together with the templates parameters are defined. The actual values for the template parameters are provided when the process definition using the *Process Definition Tool* is finalized. Thereby, the *Process Definer* can select different configuration options that impact the specific implementation of the *Concrete Guidance Aspect.* By default, these configuration options are the process state map location, help file location, whether an active guidance mode should be applied or not, the decision for visualization of a hint bar, and the decision for visualization of the process bar. Once the parameter values are defined, the *Aspect Generator* generates a concrete aspect that is a specialization of the *Abstract Guidance Aspect*. The dialog menu for generating the concrete guidance aspect is invoked from the File menu in the *Process Definer Tool.* A snapshot of the tool is given in Fig. 10.

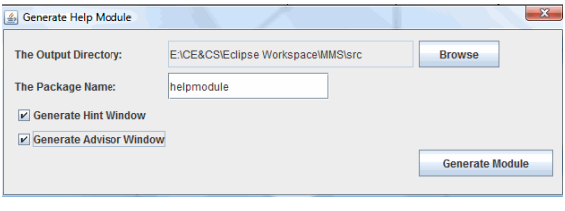

Fig. 10. Dialog for Generating Guidance Aspect

An example of concrete aspect that is generated from the abstract aspect is shown in Fig. 11. Here line 11 defines the concrete pointcut `mainWindowInitialized()` that captures the annotated constructor of the application. In line 14, the concrete pointcut `eventTriggered(Event event)` captures the annotated events that lead to state transitions in the process. The methods `initializeMapInstance()`,

`initializeModule()` and `updateModule()` define the concrete implementation for accessing the process state map and help files, the initialization of the help modules and the rendering of the help messages, respectively.

```
1.  package helpmodule;
2.  import java.awt.Window;
3.  import helpmodule.processstatemap.ProcessStateMap;
4.  import helpmodule.ui.*;

5.  public aspect MMS-Guidance extends GuidanceAspect{
6.  Window mainWindow;
7.  HelpUIComponent hintUIComponent;
8.  HelpUIComponent advisorUIComponent;
9.  HelpFileAccessor helpAccessor;
10.
11. pointcut mainWindowInitialized():
12. call(*.*.new(..)) && @annotation(ApplicationInitialization);
13.
14. pointcut eventTriggered(Event event) :
       ( execution(*.*.new(..)) || execution(* *(..)) ) && @annotation(event);
15.
16. public void updateModule()
17. {  hintUIComponent.updateHelpInformation();
    advisorUIComponent.updateHelpInformation();    }
18.
19. public void initializeMapInstance()
20. {  map =
    ProcessStateMap.readSerializedInstance("src/helpmodule/resources/map.xml");  }
21.
22. public void initializeModule(Window mainWindow)
23. {  this.mainWindow = mainWindow;
24.
25. helpAccessor = new HelpFileAccessor("helpmodule.resources");

26. hintUIComponent = new HintWindow(map, helpAccessor);
27. hintUIComponent.updateBounds(mainWindow.getBounds());
28. mainWindow.addComponentListener(new MainWindowListener(hintUIComponent));

29. advisorUIComponent = new AdvisorWindow(map, helpAccessor);
30. advisorUIComponent.updateBounds(mainWindow.getBounds());
31. mainWindow.addComponentListener(new MainWindowListener(advisorUIComponent));

32. updateModule();   }}
```

Fig. 11. Generated Concrete Guidance Aspect

In general, the provided configuration options and the generation of a concrete aspect might be sufficient. However, if needed, after generation of the concrete guidance aspect, it is also possible to modify and as such further customize the aspect manually. Here, we consider a three layered inheritance approach which is illustrated in Fig. 9. We have given an example in which the generated aspect *Concrete Generated Guidance Aspect-MMS* is further manually extended by *Concrete Manually Modified Guidance Aspect-MMS.*

It should be noted that the abstract *Guidance Aspect* is reusable for all applications in the project that require help. Depending on the nature of each application, the organization might reuse the aspect generator to produce the default help modules or extend either the generated aspect or the abstract aspect to further customize the aspect.

5.2 End-User Experience after Weaving the Aspect

When the application is initialized, the generated aspect weaves the required functionality in the application code. As a result of the weaving process, the application will be monitored for process state changes and the end-user will be guided by the necessary help. As displayed in Fig. 12, by default, the weaving process will add two modular panels to the application: *Hint Bar* on the bottom, and *Process Panel* on the right of the application window. The hint bar provides information about the current context of the application. These are the help descriptions that have been defined for the defined steps using the *Help Definition Tool* in Fig. 7. For each state change, the Guidance Aspect ensures that information is updated whenever the context of the application is changed.

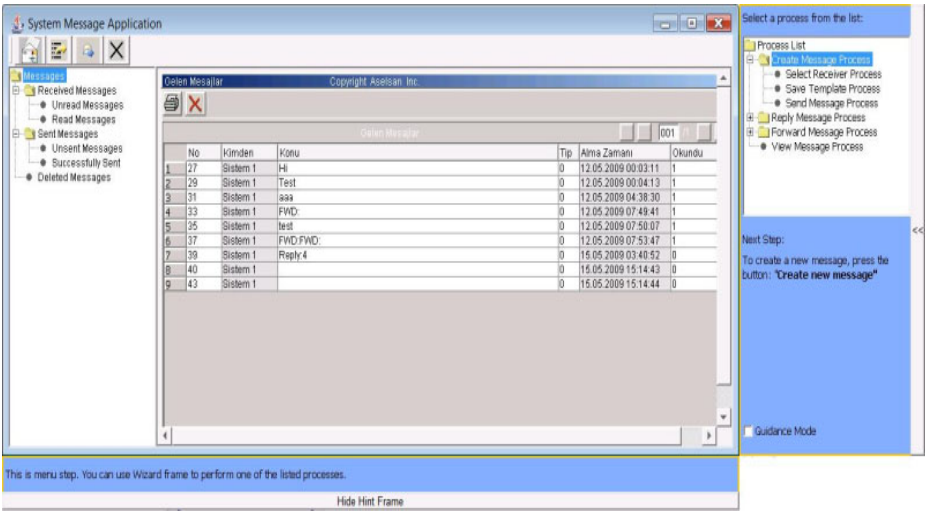

Fig. 12. MMS Application modularly extended with Embedded Help

The *Process Panel* on the right shows the process as defined before using the *Process Definition Tool* in Fig. 6. When the user selects one of the processes on the panel, the instruction for moving to the next step of the process will appear. The message that is presented represents the help description for the transitions that have been defined using the *Help Definition Tool* in Fig. 7. The tool also provides a way to activate a so-called *guidance mode* to ensure that the user can only perform the transitions that are necessary to complete the process. In particular, this guidance mode might be necessary for systems that are safety-critical, when the end-user has to follow the defined process. On the other hand, if guidance mode is not activated, both panels can also be hidden by the user.

Note that the hint bar and process panels are modularly attached to the window of the application. This ensures that no changes need to be made in the original GUI of the application. Likewise the problems of space limitations when integrating context-sensitive help have been eliminated.

The generated aspect provides the default panels which appear to be sufficient for most applications of the Aselsan's applications. However, when a dedicated hint bar or process panel is desired, this can also be relatively easily defined. In this case we have two different options. One possibility is to change the aspect generator resulting in a change of the hint bars and process panels for all the applications for which help was defined. The second option is to refine the Abstract Guidance Aspect manually. The latter option requires manual intervention but the change to the aspect is modular; we do not need to change the existing aspects.

6 Evaluation

One of the key concerns of the developers is to define the process and the related help in a cost-effective manner. To evaluate the cost of using *Assistant-Pro*, the tool has been provided to a software development group to define user assistance for the *Listing and Listening of Voice Records Application (LLVR)*. The LLVR had about the same number of process states (12 in total) and transitions (24 transitions). For each transition, a help description was provided in both English and Turkish and hence 48 help descriptions were provided. The software development group had a standard background in Computer Science and Software Engineering. The time for using the necessary activities of *Assistant-Pro* has been recorded for analysis later on. Based on the following adopted general cost model *C* for developing embedded user assistance we have provided an evaluation of *Assistant-Pro*:

$$C = C_{Learning} + C_{ProcessAnalysis} + C_{ProcessModeling} + C_{Help} + C_{Implementing\ Aspect}$$

The cost constituents of the cost model and the results of the evaluation are explained below:

- $C_{Learning}$: This is the cost for training the developers in using the tool. For this purpose, a presentation of *Assistant-Pro* has been given, which took around 2 hours together with additional discussions. Before and during the training, the developers were given a 40 page manual of *Assistant-Pro* that they could read in about 4 hours. The tool itself is quite easy to understand since the software developers have in effect to define a kind of state diagram for the possible steps. Since the notion of state diagram was well-known it was not difficult to define the states of the process and the related transitions. Altogether for training costs we can state that it took no more than one working day to train the responsible persons.
- $C_{ProcessAnalysis}$: This is the cost for analyzing the application and defining the potential set of steps and the processes that can be derived from these. In principle this is a domain analysis activity [5] that needs to be carefully done, and actually the time required to complete the activity does not differ with traditional development of embedded user assistance. It should also be stated that the analysis of the application and defining the basic steps is in effect a variable process modeling activity that is dependent on the complexity of the application. The more complex the application is, the more time it will require to define the process steps, transitions and the related help. In any case, although the process modeling

activity needs to be carefully defined it has also an explicit benefit that the developer is forced to think about the process. In fact, the reflection on the process and the modeling supports the reduction of faults that could be integrated in the application if the process is not explicitly defined as is the case in conventional user assistance design. For the process analysis it took around 3 hours to define the steps and the transitions among these steps for the LLVR case.

- $C_{ProcessModeling}$: This is the cost for defining the steps and transitions after the process analysis phase. In fact defining the steps is quite straightforward and consists of adding steps, transitions and processes in the *Process Definition Tool*. As such it took for the developers not more than 2 hours to define the process for the LLVR case.
- C_{Help} : This is the cost for defining the help for the steps and the transitions. This is also a variable cost and depends on the number of steps and transitions in the process that needs guidance. For the LLVR case, it took around 2 and a half hours to define the related help to the steps and transitions. In practice, only simple, one-sentence-based text phrases were defined.
- $C_{Implementing\ Aspect}$: A final cost of the tool is the implementation of the aspect which is dependent on whether the aspect is generated or (manually) extended. In case of generation, the Guidance Aspect is generated in the *Process Definition Tool* and this has a largely negligible cost. In case of extension the cost is related to overriding the abstract pointcuts and methods, and for this it is required that the developer has a basic knowledge of aspect-oriented programming. In our case, the time for defining the pointcut and integrating (weaving) the aspect in the application took about 3 hours (by an AOP programmer).

The cost model and the related reasoning supported the business case for transitioning to an embedded user-assistance approach using *Assistant-Pro*. Aselsan selected the two example applications that we have described in this paper as pilot studies. Based on the evaluation done by Aselsan using the above cost model, evaluation of both case studies was very positive.

7 Discussion and Lessons Learned

The development of embedded user assistance is not trivial and this is even further complicated if context-sensitive assistance is required. One of the key obstacles is certainly the crosscutting nature of the help concerns. The tool *Assistant-Pro* that we have discussed is implemented within the industrial context of Aselsan.

Initially, Aselsan adopted the traditional cycle of developing user assistance (non-embedded) which had several obstacles. The nature of the applications that are developed by Aselsan (process-sensitive, safety-critical) defined the key motivations to develop an improved embedded user assistance system that could track the process, and be reusable for a broad set of applications. The company already had mature knowledge on user assistance systems and experienced maintenance problems in the regeneration and updating of user manuals for multiple applications in case of changes to the process or the application. Although the main cause of the required maintenance activities seemed to be the crosscutting nature of guidance concerns, the decision for an aspect-oriented solution, however, was not straightforward. In fact

deciding for a new user assistance development approach would mean that the (user assistance development) processes at Aselsan had to be redefined. On the other hand the topic of aspect-oriented software development was already known to the company because of earlier consultancy and education activities in this domain. Together with the structural maintenance problems and a possible promising solution it was decided to analyze the aspect-oriented design solution and based on this, define two pilot projects as to evaluate the approach as proposed by *Assistant-Pro.*

The results of the two pilot studies were very positively evaluated both by the upper level management and the senior development team. In fact *Assistant-Pro* appeared to have several benefits for the organization that we could categorize as general advantages of embedded user assistance systems, and additional, specific advantages related to the adopted approach in Assistant-Pro.

General Advantages

- *Fewer faults*

The cost for user assistance development is not only reduced due to the integrated help, but also as a result of explicit, context-based guidance. In this way users avoid making faults, and as such the costs related to usage of the tools, and the costs that would be derived from wrong usage are further reduced. Using the active guidance mode in *Assistant-Pro,* the need for repeating tasks several times or moving back and forth over the process to obtain the best results is eliminated.

- *Communication between User Assistance Specialist and Developer*

In conventional user assistance development, the tasks of the user assistance specialist and the developers are strictly separated and there is less communication among both. The lack of communication between the user assistant specialist and the developers lead to problems in defining and aligning the appropriate help for the process states. In embedded user assistance, this problem is eliminated because a close cooperation between the *User Assistance Specialist* and the software developer is required. The *User Assistance Specialist* should normally be the person to design the layout of the user assistance and to write all the content, whereas the software developer will mainly be responsible for the technical implementation. Similarly, *Assistant-Pro* provides a common platform for communication between the *User Assistance Specialist* and *Developer.* One could argue that the introduction of explicit communication channels will also add additional costs to the organization. This is however not a problem of our approach, but of embedded user assistance in general.

Specific Advantages

- *Enhanced support for End-Users*

For the end-users it is important that assistance is given at the right time and right place in the application. With *Assistant-Pro,* help can be relatively easily defined based on the context. We have focused on the process context and as such explicitly modeled the process. Early validation of the tool *Assistant-Pro s*hows promising results. In fact the information on the process that was earlier provided in PDF documents or Wiki pages is now defined in the *Assistant-Pro Help Definer Tool,* and later

on delivered as modular help as shown in Fig. 12. With the application the end-user can even be strictly guided through the process, where faulty steps are prevented.

- *Modular and Reusable Guidance*

Although we are developing *embedded* user assistance, the help concern is not embedded in the application but modularly integrated. In case of required changes to the help concern this can be easily located and adapted to the new needs by updating the aspects. All the information is stored in XML files and aspects can access these files to define embedded user assistance. Through defining aspects and automating weaving of help concerns in the final application, the complexity of the help concerns is also reduced and maintenance is substantially improved with respect to the earlier process followed. The aspect generator and the generated guidance aspects can be reused for multiple applications.

- *Solving the Space Problem because of Modular UI Extension*

Not only the system is designed in a modular way but also the panes for rendering the help, such as hint bar and process bar, are modularly extended to the original window of the application. Although the user assistance is integrated in the application it does not occupy the space of the original application. Because of this modular UI extension we have solved the space problem that relates to embedded user assistance systems in general. In general this is considered as one of the important problems when developing embedded user assistance systems.

- *Integration with legacy code*

Many applications are not developed from scratch but reside in the project as legacy code that cannot be easily removed. Most of this legacy code however still requires guidance. Since aspects are used to define help, it is now also possible to define help for legacy code. This was very hard if not impossible in earlier approaches.

- *Overall Cost of Help Documentation, Training, and Usage*

In the former approach besides creating the manuals, an additional cost was required for the stakeholders to understand and use the guidance documents. In case the process and the related documentation was not clear or too complex, this soon resulted in the demand for extra training, leading to additional, often unforeseen costs. With *Assistant-Pro* the documentation is not separate but part of the application. Assistance is provided to the user whenever necessary. In fact, because of the active guidance definition, we can state that the training is defined whenever the user starts using the tools. This situation leads to a reduced number of demands for training and as such a reduced overall cost related to user assistance for the application in the organization.

8 Related Work

Two of the main distinguishing characteristics of Aspect-Oriented Programming (AOP) systems is that they allow quantified programmatic assertions over programs written by programmers oblivious to such assertions [12]. In our approach we have

used an annotation based approach and as such somehow explicitly declared the points that we would like to give advice to. A key reason for adopting annotation-based AOP is to define expressive and robust pointcuts [17][22][25]. Alternatives to annotation-based pointcut descriptions very often select joinpoints based on the signatures of language elements based on naming conventions or structural patterns. It has been shown that such pointcut descriptions are more fragile and either fail to catch the required joinpoints or catch undesired joinpoints. Annotations can provide semantic information that are attached to the required places in the code and as such can support the definition of robust pointcuts. In our case we have to deal with user assistance for process-sensitive systems, which are very often mission-critical in defense applications. Hereby, the definition of robust pointcuts is a strict requirement and it is not acceptable that either the wrong process element is caught or somehow a process element is missed due to an inappropriate pointcut description.

In our approach, process guidance consists of modeling the process that needs to be adopted and the corresponding help that is defined for the process steps. Process modeling has been addressed in different domains including method engineering [6][19][29] and meta-modeling [33]. A particular approach for process modeling is *situational method engineering* which is targeted at creating method components rather than complete methodologies and tailoring them to specific situations or context at hand [28]. Several efforts have been made to standardize situational method engineering, which has led to, for example, the OPEN Process Framework (OPF) [26][37], OMG's Software Process Engineering Metamodel (SPEM) [32], and ISO/IEC 24744 [21]. These proposals have been used for modeling methods for different purposes but we do not know of any approach that model the process for providing active user assistance. In our project, the context for the process, or methodology that needs to be followed, was primarily defined by the various command and control applications. In general, situational method engineering and process modeling approaches provide a broader set of method fragment types than that we have defined. But the limitations of the method fragment types was a deliberate design decision since in the project it was neither necessary nor required to define a detailed set of method fragments. The project constraints required that the process modeling had to be as simple as possible to decrease the learning curve and increase its practical use for the average engineer. Nevertheless, we think that the presented approach is general and, if required, the process metamodel could be enhanced with new method fragment types to support the user assistance approach.

For implementing the process guidance we have adopted an AOP approach together with a process modeling approach. The adoption of the techniques that we used was mainly defined by the industrial context of the project. Most projects that needed help were developed in object-oriented languages and there was a mature developer expertise in this context. In fact, for monitoring the process steps we have also analyzed the application of object-oriented design patterns. For example the *Observer* pattern (also known as *Publish-Subscribe*, or *Listener*) is particularly useful for graphical applications. Using this pattern one or more objects (the subscribers) register their interest in being notified of changes to another object (the publisher). In our case we might have defined a *GenericListener* (instead of *GenericAspect)* and register it with each GUI element. Unfortunately, as it has been described in the literature [15][18][24], the invasive nature of pattern implementations and the scattering and

tangling of such an implementation with the base code may reduce the modularity of the system, thus affecting its understandability and maintainability. Compared to patterns the AOP solution in our case seemed also to be more powerful. First of all, by using an aspect instead of a pattern we avoided the necessary refactorings that were required to provide guidance functionality to the existing code. For several legacy applications that required guidance, refactoring was very hard or not possible. Moreover, using an aspect we could be more selective in providing guidance to the process actions. Process definers could easily experiment with different pointcut descriptions to define the appropriate guidance. Further, the guidance aspect was only woven into the selected GUI components.

9 Conclusions

In this paper we have presented our experiences in an industrial project for defining an aspect-oriented tool framework for context-sensitive embedded user assistance. The problems were derived directly from the context of the industrial projects of Aselsan in which multiple applications are developed that require guidance for the users with respect to the state of the process. Unfortunately, user assistance so far was largely defined using conventional user assistance techniques that define help separately from the application. The problems related to both, the end-user perspective and developer perspective, have clearly been encountered by Aselsan. In particular we have focused on the problem of the crosscutting nature of help concerns, reuse across multiple applications, lack of explicit process and the necessity for tool support. Existing approaches failed to provide a solution to at least one of these problems. Although existing approaches can be sufficient for defining help when we consider the definition of relatively simple help support such as tool-tips, defining help based on the process-context did not seem trivial. Usually the defined solutions are too static and cannot easily track the changes of the process state and provide the related help at run-time.

To address the problems we have developed the aspect-oriented tool framework *Assistant-Pro* for defining embedded user assistance for guiding the end user with respect to the process state. One of the key problems in defining embedded user assistance is the crosscutting nature of help concerns and to solve this problem, the need for aspect-oriented techniques was necessary. As such in the framework we have modularized and implemented help concerns as aspects. In principle, we did not encounter any difficult problems in doing this. Moreover, we were also able to reuse the aspects for defining different help concerns. For tackling the issue of guidance with respect to a process, we have provided a model that integrates both process modeling and guidance concepts. The conceptual model has been realized in the framework and is used by the tools of *Assistant-Pro.* Modularization of the guidance concern into an aspect here is the driving solution, but it should be noted that the aspect itself is also loosely coupled from the process state and the help definitions. The strength and novelty of our solution is not in the isolated parts but in the integration and the coordination of the different parts.

The tool has been evaluated with respect to a cost model that we have defined and the concerns of the relevant stakeholder. Our analysis shows firstly that the tool can

be used by the developers with relatively little effort to develop the process model and the related user assistance systems. For the end users, the key advantage is that the help is now embedded in the system, so that they do not waste time when help is needed. This reduces the cost for additional training about usage of the tools. Further, since end users are explicitly guided by the predefined process, the chance of making faults is substantially reduced.

In fact the framework as such is complete and is tested for different applications. In our future work we will continue the technology transfer activities and apply the framework for a broader set of applications.

Acknowledgments

We would like to thank the anonymous reviewers for their extensive and very useful feedback. Different persons have participated in discussions throughout the Embedded User Assistance project. We would like to thank Alper Bostancı, Baki Demirel, and Özgü Özköse Erdoğan from Aselsan-REHIS group, and Feyyaz Ertugrul and Mert Özkaya from Bilkent University.

References

1. Aselsan, http://www.aselsan.com.tr/default.asp?lang=en (accessed March 2010)
2. AspectJ Development Tools (AJDT), http://www.eclipse.org/ajdt/ (accessed on September 2009)
3. Ames, A.: Just what they need, just when they need it: An introduction to embedded assistance. In: Proceedings of the 19th Annual International Conference on Computer Documentation, pp. 111–115. ACM Press, New York (2001)
4. Andrade, O.D., Novick, D.G.: Expressing help at appropriate levels. In: Proceedings of the 26th Annual ACM International Conference on Design of Communication, Lisbon, Portugal, September 22-24 (2008)
5. Beyer, H., Holtzblatt, K.: Contextual Design: A Customer-Centered Approach to Systems Designs. Morgan Kaufmann, San Francisco (1997)
6. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology 38(4), 275–280 (1996)
7. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Reading (2000)
8. Elrad, T., Fillman, R., Bader, A.: Aspect-Oriented Programming. Communication of the ACM 44(10) (October 2001)
9. Ellison, M.: Embedded user assistance: The future for software help? Interactions 14(1), 30–31 (2007)
10. Fayad, M.E., Schmidt, D.: Object-Oriented Application Frameworks. Communication of the ACM 40(10) (October 1997)
11. Fast-Help, http://www.fast-help.com/ (accessed on September 2009)
12. Filman, R., Friedman, D.: Aspect-oriented programming is quantification and Obliviousness. In: Proc. of the Workshop on Advanced Separation of Concerns, in conjunction with OOPSLA (2000)

13. Galitz, W.O.: The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques. Wiley, Chichester (2007)
14. Gamma, E., et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
15. Garcia, A.F., Sant'Anna, C., Figueiredo, E., Kulesza, U., Pereira de Lucena, C.J., von Staa, A.: Modularizing design patterns with aspects: a quantitative study. In: AOSD, pp. 3–14 (2005)
16. Grayling, T.: If we build it, will they come? A usability test of two browser-based embedded help systems. Technical Communication 49(2), 193–209 (2002)
17. Gybels, K., Brichau, J.: Arranging language features for patternbased crosscuts. In: Aksit, M. (ed.) Proc. of 2nd Int' Conf. on Aspect-Oriented Software Development, pp. 60–69 (March 2003)
18. Hannemann, J., Kiczales, G.: Design Patterns Implementation in Java and AspectJ. In: Proc. of Object Oriented Programming Systems Languages and Applications 2002 (OOPSLA 2002), pp. 161–173 (November 2002)
19. Henderson-Sellers, B., France, R., Georg, G., Reddy, R.: A method engineering approach to developing aspect-oriented modeling processes based on the OPEN process framework. Information and Software Technology 49(7), 761–773 (2007)
20. Ter Hofstede, A.H.M., Verhoef, T.F.: On the feasibility of situational method engineering. Information Systems 22, 401–422 (1997)
21. ISO/IEC. ISO/IEC 24744, Software Engineering – Metamodel for Development Methodologies, 1st edn. (2007)
22. Laddad, R.: AOP and metadata: A perfect match. In: AOP@Work Series, IBM Technical Library (March 2005)
23. Miller, M.G.: Writing User Documentation: Hints For Document Writers, 2nd edn., CreateSpace (2009)
24. Monteiro, M.P., Fernandes, J.M.: Towards a catalog of Aspect Oriented Refactorings. In: Proc. of Aspect Oriented Software Developmnet, AOSD 2005 (2005)
25. Nagy, I., Bergmans, L., Havinga, W., Aksit, M.: Utilizing design information in aspect-oriented programming. In: Proceedings of International Conference Net.ObjectDays, NODe2005, Gesellschaft für Informatik, GI. Lecture Notes in Informatics (2005)
26. OPEN Process Framework (OPF) Web Site, `http://www.opfro.org/`
27. Ray, D.S., Ray, E.J.: Embedded help: Background and applications for technical communicators. Technical Communication 48(1), 105–115 (2001)
28. Ralyté, J., Rolland, C.: An assembly process model for method engineering. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 267–283. Springer, Heidelberg (2001)
29. Rolland, C., Prakash, N., Benjamen, A.: A multi-model view of process modelling. Requirements Eng. J. 4(4), 169–187 (1999)
30. Sleeter, M.E.: Online help systems: Technological evolution or revolution? In: Proceedings of the 14th Annual International Conference on Computer Documentation Conference, pp. 87–94. ACM Press, Research Triangle Park (1996)
31. Smith, D.: Developing online application help. Hewlett Packard Journal 45(2), 90–95 (1994)
32. Software Process Engineering Metamodel Specification, Object Management Group Inc. (2006), `http://www.omg.org/technology/documents/formal/spem.htm`

33. Stahl, T., Voelter, M.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006)
34. Tidwell, J.: Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media, Sebastopol (2007)
35. Turk, K.L., Nichols, M.C.: Online help systems: Technological evolution or revolution? In: Proceedings of the 14th Annual International Conference on Computer Documentation Conference, pp. 239–242. ACM Press, New York (1996)
36. Weber, J.H.: Is the Help Helpful? How to Create Online Help That Meets Your Users' Needs. Hentzenwerke Publishing (2004)
37. Zowghi1, D., Firesmith, D.G., Henderson-Sellers, B.: Using the OPEN Process Framework to Produce a Situation-Specific Requirements Engineering Method. In: Proceedings of SREP 2005, pp. 29–30 (2005)