Integrating EDEN and Knowledge-Based Active Help Systems

Roberto Silveira Silva Filho
Department of Information and Computer Science
University of California, Irvine
rsilvafi@ics.uci.edu

Abstract

High Functionality Applications (HFA) are complex and powerful applications that provide a large set of features and resources to the users. Active help systems are used to facilitate the use of these applications. This paper describes the project of an extension to the EDEM system in order to integrate this application with a knowledge-based system for active help systems. It is described the integration of the EDEM system with a project of a generic API that converts low-level interface events into higher level observational variables. This API is used as the basis for the implementation of knowledge-based active help systems. This project lies in the software architecture proposed by Microsoft Research Lumière project. A system to support the implementation of context-sensitive help based on the runtime analysis of user's goals and intentions.

Keywords*: Knowledge-based systems, User modeling, Smart help agents, Usability, EDEM .*

1. Introduction

This paper describes the project of a generic API that converts low-level interface events into higher level observational variables, used by a knowledge-based active help system. This API is specified in the context of a software architecture that support the implementation of context-sensitive help based on the runtime analysis of user's goals and intentions.

The EDEM system [Hilbert99] is used as the basic system for collection of user interface events. The necessary modifications of this system to continuously collect interface events is described. These low-level events are used to infer high-level goals from the uses of an application. The Lumière project architecture is used as the context in which these data will be used. This application is intended to provide runtime assistance to software users, based on their expertise level, their goals, and the different usability requirements of the application. The determination of the user's goals, the current user intent and the appropriate time to provide a help information are the most important challenges in the implementation of this system. The requirements and lessons learned during the development of this prototype are used as the basis of our work.

1.1. Paper Description

This paper is organized according to the following structure. Section 2 presents the motivation of our work, section 3 introduces the concept of knowledge-based systems. In section 4, the Lumière project is described in detail, section 5 presents an overview of Bayesian networks. Section 6 describes the EDEM system with its characteristics, section 7 describes the project of the adaptation of EDEM with the Lumière architecture and, finally, in section 8 some future work is discussed.

2. Motivation

The Human Computer Interaction (HCI) field is concerned with the improvement of the way people use computers to work, think, communicate, learn, critique, explain, argue, debate, observe, decide and so on [Fischer00]. In this context, the development of applications that improve this interaction is a challenge.

High Functionality Applications (HFA) are complex and powerful applications that provide a large set of features and resources to the users. They are usually designed to serve the needs of a large and diverse user population. These applications are difficult to use at first, but with time and experience, allows the users to perform a big set of tasks. The design of interfaces for these systems is a big challenge. The design of these systems, focusing on the novice user, can hinder the use of all the resources of these applications. On the other hand, the design of the system tailored to the expert users, can compromise the usability and learnability of these systems [Fischer00, Fishcher99].

3. Knowledge-Based Systems

An approach to solve this trade-off is to provide the applications with means to adapt its behavior according to the expertise of the user. In order to provide this ability to the applications, the computer must be provided with a considerable body of knowledge about the problem domains, the communication process, and the agents involved. Fisher [Fischer00] characterizes three different types of user knowledge as follows. Applications that use this knowledge approach are known as knowledge-based systems.

Knowledge about the problem domain. In order to be able to determine the goal of the user, the system must have knowledge about the capabilities of the domain that it automates. This knowledge constraints the possible interactions with the application. For example, the drawing of a square in the screen may signify a class component in a CASE tool, a simple picture in the paintbrush or a specific component in a CAD for electronics.

Knowledge about the communication process. The interaction between the user and the system should be customizable. An important issue in the automated user assistance is determine when and where to interrupt or provide information to the user. Horvitz [Horvitz99] presents many facts that must be considered in the determination of the right moment to interact with the user. In general, these features must be adaptable to the expertise level of the user, also being able to be turned on and off.

Knowledge about the communication agent (user). In order to collaborate with the user in his exploration and use of the interface, the system must be able to figure out what the user really knows and does. According to Fisher, "Simple classification schemes based on stereotypes such as novice, intermediate or expert users, are inadequate for knowledge-based systems because these attributes become dependent on a particular context rather that applying to users globally".

Hence, the system must be able to adapt, at runtime, to the different needs of the users in order to be able to say the "right" thing at the "right" time, and in the "right" way. To say the right thing is to provide the relevant information to the current context of the user activity. To say things in the right time requires to balance between the costs of intrusive interruptions and the lost of context-sensitivity of deferred alerts [HBHHR98]. To say things in the right way is especially important for users that have disabilities.

4. The Lumière Project In Detail

The Lumière project [HBHHR98] centers on integrating probability models, in this case Bayesian networks, with an active help system to provide runtime assistance to software users. The Bayesian user models are used to infer the user's needs and goals based on models that represent and reason about the user's background, actions and queries.

The main challenges of this project were: (1) The construction of the Bayesian user model. (2) The implementation of the reasoning mechanism about time varying goals of the users, based on their observed actions and queries. (3) The instrumentation of a system to collect user interface events from the applications in use. (4) The development of a language that converts low-level interface events into higher level observational variables, represented in Bayesian user models. (5) The development of persistent user profiles to store and represent the evolution and the expertise of each user. (6) And the development of a general architecture that integrates all these components in an active help system.

These main challenges are described in detail in the next sessions. The following sessions summarize the goals and development experience of the authors of the Lumière project, described in [HBHHR98].

4.1. Construction of the Bayesian User Model

The Bayesian networks cope with the problem of determining the user beliefs, intentions, goals, and needs, providing a statistical model to represent this knowledge. The Lumière project captures an influence diagram as shown in Figure 1. This network provides a decision model used to infer the user's goals and needs.

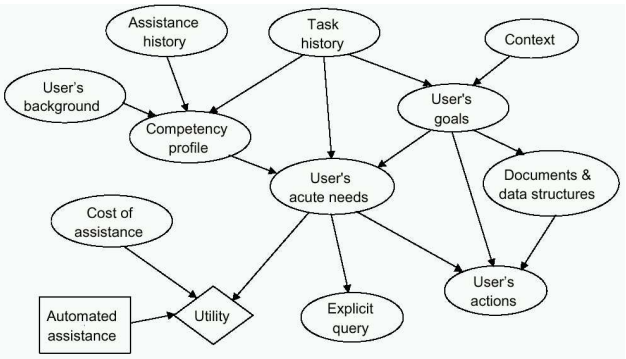

Figure 1 The Lumière Influence Diagram used to provide statistic inference about user's background, goals, and competency. [HBHHR98 - (Figure 1)].

The balloons in the diagram represent variables, while the arrows express the mutual influence of these variables. In this example diagram, the user's goals influence his acute needs. The user's competence is influenced by previous background experience, previous assistance history and the history of successful accomplishment of tasks with the tool. The user's actions are also influenced by his current needs, goals and the feedback of the application in the form of documentation and data structures (this last, in the special case of spreadsheets).

The help system must provide valuable information to the user, balancing between its intrusiveness (whenever a useful not requested help is provided), and its lack of assistance, due to the probability that the user does not need the provided information. Hence, the utility of a help system, which is a function of the user's experience with the tool, must be pondered according to the nature of the action, the cost of the action and the user's needs.

In this context, **Actions** are sets of event representing the interaction of the user with the application. Eric et al. defines **Goals** as "target tasks or subtasks at the focus of a user's attention" and **Needs** as "information or automated actions that will reduce the time or effort required to achieve goals".

4.1.1. Defining the user model

The Bayesian model used for the project was created using the results collected by psychologists in the Microsoft Usability Lab, based on observation studies of the use of the target application (Microsoft Excel). The objective was to determine how accurate the human observers could be in guessing, by observing and monitoring a copy of the user's screen: (1) when the user needed assistance and (2) the type of assistance needed.

This study identified some **classes of evidence** that categorize different sets of activities a user can be performing when using an application. These classes are listed as follows.

- **Searching**: A user is performing a search when he or she perform repetitive scanning patterns, when attempting to search for, or access, an item or functionality. Some examples of such searching patterns are: the moving or the mouse around the desktop, the clicking on multiple non-active regions, or the exploration of menu options without clicking or selecting any function.
- **Focus of attention**: A user is focusing on some interface object, when it is selected or active at a given moment. Some examples of focus are: the dwelling on parts of a document or on specific controls of an interface after a search period.
- **Introspection**: Represents a sudden pause after an activity period or a decrease in the rate of interaction with the application.
- **Undesired effects**: Are characterized by attempts to return to a previous state after an action. For example, The undoing operations, or when the user closes a dialog box, just after it was open, without performing any operation offered in that context.
- **Inefficient command sequences**: These sequences are detected when a user performs operations, using a longer sequence of actions, that could be done with a shorter set of operations or with accessible shortcuts or interface elements as buttons.
- **Domain-specific syntactic and semantic content**: Comprehend all other activities that a user may be performing, that are specifically bounded to the particularities of a class of applications or a domain. For example, special interactions with the application interface, and activities that can be performed with a specific CAD application or drawing tool, characterized by special mouse strokes, movements, shortcuts and operations.

The Built of effective user models using Bayesian networks depends on defining appropriate variables and states of variables in this network. An example of a simple Bayesian network is provided in Figure 2. In this example, the likelihood that the user needs assistance is inferred based on information stored in his profile, as well as observations of recent activities (interface events). The example in Figure 2 represents a influence network, in which the influence of a pause after an activity and the likelihood that the user needs assistance can be determined.

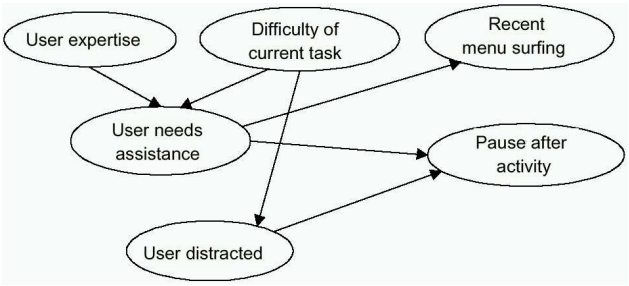

Figure 2 A subset of the complete Bayesian user model generated for the Microsoft Excel.
[HBHHR98 - (Figure 2)].

An important point in this example is that it express the fact that the user may pause if he or she finds the task difficult or is distracted by an external event that is not related to the application itself. According to this model, the probability that the user needs assistance is also shifted by the fact that he has more or less experience

with the application.

The task of building a Bayesian model is not easy and requires the evaluation of the use of the interface in many different scenarios. The network can also consider the application as a whole or it can model a specific interaction, for example, the need of help for the use of a specific menu or dialog box in the system.

4.1.2. Temporal Reasoning

The Bayesian networks described in the previous examples are not temporal, they do not consider different time slices of the network and do not evolve with time. While the user is working with the application, his interest focus can shift as well as his objectives. In order to express this temporal evolution, Eric et al. proposes an extension to that model that includes temporal dependencies between goals at a present moment and goals at a previous time.

The solution proposed for this problem was the definition of different time slices in the Bayesian network, generating a kind of 3D reticule, in which the third dimension is the time. The probability of a goal associated to the node is, in this extension, a function of the previous values of this node in time. According to Eric et al. "The intuition behind this approach is that observations, seen at increasingly earlier times in the past, have decreasing relevance to the current goals of the user". The current probability of a current goal is a function of the time passed since the last occurrence of an action, as well as the conditional probability of those actions in the history of events. As times goes by, this probability decays to the value for the case where the evidence is not observed.

This model uses the concept of evidential horizon and decay of variables:

- The **evidential horizon** is the number of actions, or the amount of time in which a probabilistic relationships persist without change.
- The **decay variables** specify a functional class (linear, exponential, quadratic, and so on) and the parameter defining the details of how the conditional probabilities change, after a horizon is reached. The decay can be expressed as a function of the time elapsed since the last action, or as a function of the number of actions occurred since the observation was noted to become true.

4.2. Collecting user interface events from the applications

In order to establish a link between user actions and interface events, a stream of user actions (interface events) need to be collected and analyzed. For the Lumière project, a special version of Excel, with the ability to capture interface events was defined. The events captured were: mouse events, visit to menus, the open and close of dialog boxes, selection of specific objects as drawing objects, charts, cells, rows and columns. These events are stored in an event queue with their respective timestamps. In this stage they are called **observations**. This queue can be searched by the applications in order to infer and compose higher-level events.

4.3. Development of a language that converts low-level interface events into higher level observational variables represented in Bayesian user models

In order to convert this low-level user interface events stream in higher-level predicates or events representing user actions, a set of operations to this event queue were defined. These operations were combined, originating the Lumière Event Language. This language allows the combination of observations in sets and Boolean expressions. It also allows the arbitrary composition of these high-level events, with observations to form new events. This language also allows the specification of filters that can be combined and integrated in the system.

The following primitives are provided by the Lumiére Event Language API.

- **Rate(x_i, t):** Returns the number of times an atomic event x_i occurs in t seconds or t commands.
- **Oneof({$x_1 \ldots x_n$},t):** Returns true if at least one event in the provided set occurred in the time period t.
- **All({$x_1 \ldots x_n$},t):** All events of a denoted set of events occur at once in any sequence within t.
- **Seq({$x_1 \ldots x_n$},t):** Events occur in a special order within t.
- **TightSeq({$x_1 \ldots x_n$},t):** Events occur in a special order within t and no other events occur.
- **Dwell(t):** There is no user action for at least t seconds.

These primitives, combined with low-level events may be used to compose filter expressions such as "a user dwelled for at least t seconds following a scroll". This language also allows the capture of keystrokes in order to interpret a 'ctrl+S' command or a 'File/Save' menu operation as the same event. The *time scale* used in these expressions can also be adjusted to express the speed the user provides commands to the system, this is feature is called **scaled time**.

The events collected by the filters are used to feed the

temporal Bayesian network, which reasons about a set of problems. In the Lumière prototype, there were identified and implemented 40 problems, associated with 40 areas of assistance.

The system also uses a free-text help query module to get direct queries from the user. This information is feed to the Bayesian inference model in order to enhance the cues about the current needs of the user.

4.4. Development of a general architecture that integrates these components

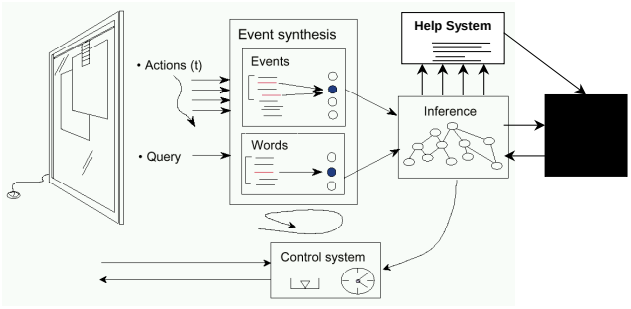

Figure 3 The Lumière Software Architecture

The overall Lumière/Excel architecture is described in Figure 3. Events captured from the interface are transformed into timestamped observations. The observations are input to the Bayesian model, and the probability concerning user needs is inferred. The system also infers the likelihood that a user needs assistance at a given moment. This probability is used to decide when to start the smart agent that provides the appropriate help to the user, in a spontaneous way. A user-specific probability threshold is also specified, and controls the activation of the autonomous assistance.

Events are constantly collected in the application interface. The reasoning process however is performed periodically. There are two basic approaches to start the process of reasoning about the user goals. In **a pulsed strategy**, the system periodically performs a call that starts a cycle of the reasoning process, which consists on the collection of events, the recalculation of the probabilities in the Bayesian network, and the eventual feedback to the user. An **event centered approach** selects specific events to trigger this whole process. A **hybrid approach** can also be specified in which the process is run at each interval or when some important event comes up.

4.5. Development of persistent user profiles that store the evolution and the expertise of each user

The user can also customize his profile providing his level of expertise, as well as other information to customize the activity of the agent. The user profile is also augmented and updated, periodically, as the user ask for help or is helped by the agent. When the user complete tasks recognized or pointed by the agent, his level of expertise is automatically updated. A **competence profile** of the user is composed using the information about the accomplishment of key tasks and the review of help topics. Special **competency variables**, belonging to the Bayesian model, can also be customized or automatically updated by the system, using this procedure. These variables are persistently stored in the user profile.

The data from the user profile is combined with the Bayesian network in order to infer the probabilities associated with all possible user needs at the moment, and the probability that the user will need help immediately. This information is also checked against the user threshold in order to decide whether to provide the help information or not. The objective is to prevent the intrusive operation of the agent.

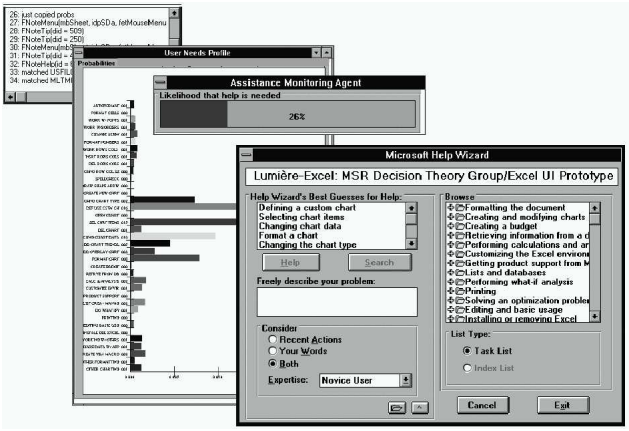

Figure 4 Inference Behind the Scenes ([HBHHR98 - Figure 6])

The Figure 4 presents a graphical representation of the results generated by this approach. The first window, on top left, display the events generated by the interface, the next window, with the horizontal bars, present the probabilities inferred by the model, that the user needs help in one of the 40 help categories available, at the moment. The percent bar window displays the user threshold. The right most bottom window displays a dialog box in which the user can customize his expertise level. It also provides a “best guesses” list of help assistance topics sorted by

the likelihood, inferred by the system, that the user needs each information. The user can also provide free-text queries that are used to get information about his needs and narrow the offer of help assistance.

Non Intrusiveness: The user can customize, through a "volume level" control, the threshold used by the smart agent to determine when to appear and provide the help information. If after the announcement of the help availability, the user does not consult the agent, it automatically disappears.

4.6. Beyond Real-Time Assistance

Based on the observation of the user actions during the use of the system, some other features were implemented in the Lumière prototype.

4.6.1. Further Reading Recommendation

As the user works with the system, the smart agent infers and stores information about the tasks the user accomplished. Based on the expertise level demonstrated during these tasks, at the end of the application, the system provides a list of recommended help topics to be reviewed offline. This offline information can include help tutorials or chapters in the manual. The non optimal use of the system features for some tasks can be considered in the identification of specific help assistance topics, in which the user did not demonstrate expertise. This information is translated in a list of recommended topics.

This list is compiled based on the following algorithm. It considers, for each area of assistance that the user did not reviewed during a session, the number of times that the probabilities of user needs for that area exceed a predefined probability threshold. Each time this threshold was transposed, the help topic is added to the list.

4.7. Simplifications

The actual help agent used by Microsoft Office 97 suite is a simplified version of this prototype. This version does not provide a persistent profile for the user, does not use rich combination of events over time and considers only a reduced set of user actions in its reasoning. The event queue is also limited, considering only a small length of events. The analysis of words and events is not performed in an integrated way. The results of the inference are available only when the user requests assistance explicitly.

4.8. A Simpler Approach

A more simple approach for the use Bayesian network models would be the explicit asking of the user about his goals. This could be done using a questionnaire that could to be filled by the user. This could be provided at the beginning of an application session and as an option in the system menu. This questioner could be made persistent and being updated by the user whenever a new task is performed. Moreover, a set of pre-defined settings (questionnaires) could be provided, as well as the ability of generating new goals through these questionnaires. This idea is similar to the customization button provided by the Argo UML Case tool [RR00].

Linton et al. [[LJS99]] proposes the development of a user model based on the observation of the interaction of a set of users with the application. This approach is based on the specification of an average user based on the statistical analysis of observations of many users performing a predefined set of tasks using the application interface. This model is used to create stereotypes and a set of basic steps that represent each interaction with the system. It is a simple approach that requires long periods of observation with a big set of users.

In [Kobsa93], many toolkits and shells to support user modeling are presented. In general, these models are based in the idea of stereotypes that allow the representation of average stereotypes as experts, novice and intermediate. The user can customize the application, selecting his expertise level. These expertise levels are based on information collected during previous analysis of the application that will use the model and, in some cases, the intuition of the developers. [Kobsa00] updates the previous classification of systems, surveying some new approaches to user modeling based in distributed shells, agents and databases.

5. Bayesian Networks

According to [Hedberg98], "Bayesian Networks are graphical models of relationships and dependencies between variables, including probabilities." This probabilistic model allows users to express "belief networks" of complex cause-effect relationships between nodes (variables). These relations are weighted to express how nodes affect each other, representing a way to express previous knowledge about the system being modeled. These knowledge is coded as dynamic and static properties of the nodes. The Bayesian networks also allow the expression of complex non-deterministic relations between nodes, allowing the inference of "might" relations, for example, the probability of a complex event to occur, based on the state of other variables in the model.

A formal description of Bayesian networks is described by Heckerman et al. [HGC95].

The Microsoft Research provides a free Bayesian Network Tool, the Microsoft Belief Network. It is freely available for noncommercial research and academic organizations in the site [MSBN]. It is available an Application and a DLL. This last provides an API that can be integrated in other applications as the one whose project we propose in this paper. This tool is programmed using the XML Belief Network File Format [XBNFF].

6. EDEM

The EDEM [HR98a, HR98b, HR98c, Hilbert99] is a system that monitors and collects UI (User Interface) events in Java applications and automatically deploys this information to usability databases using the Internet.

This system was primarily used as a support tool for software development. In special, the analysis of user interfaces usability. It uses an automatic usage monitoring usability technique called data collection. In this technique, the usability data is collected for further analysis and interpretation. The study of this data has been used to help in the detection of interface faults and future improvements of the system.

6.1. Usability Information Collection, the EDEM Approach

During the development of software user interfaces, the gulf existing between the usage expectations of the developers and the actual interpretation of the interface by the final users is a big problem. This gulf must be minimized in order to improve the usability of the application.

Many approaches exist in order to collect this information. Classical usability data is collected by the observation of the user and its interaction with the application. It is limited to small-scale tests, in which usability data are collected and reported manually by the beta test users. The EDEM approach allows the collection of these data through the Internet, in a no-intrusive and scalable way.

During the development process of user interfaces and their applications, designers project the system keeping in mind a set of usage expectations. **Usage expectations** are the developers expectations about how users and the application operational environments will behave. They are based on the domain of the application, the requirements of the software, the developers experience, and so on.

6.2. EDEM Events

The EDEM system uses the concept of usage expectations to define **Expectation Agents**. Agents are mobile pieces of code that monitor the system usage and perform actions when the user expectations they represent are violated. The expectations are represented as sets of events and logical expressions, associated to these events, that should be satisfied.

Whenever an expectation is violated, an agent can be configured to perform numerous **actions**. Such actions can range from: notify the user and/or developer of usage mismatches, report system state and/or event history for debugging purposes, provide guidance or suggestions to the user, or collect direct feedback information. In general, an EDEM server is used to store agent data reports for later usability analysis.

According to [HR98a], the EDEM **activity space** "is comprised of the objects and activities of interest in the system being monitored". They are system-specific actions logged by the EDEM system, representing a tracing of the interface events generated by the program being monitored. The **event space** "is characterized by events and entities that correspond to objects and activities in the activity space". They represent the actions being monitored. The event space is system independent and has its own name space. Events may be inferred or computed by compilation of information about activities.

6.3. Edem Components

The main components and concepts used in the EDEM system are described as follows.

- **Probes.** Probes are elements of the EDEM architecture that collect (observe) the system events (event space) and transform this information into notifications (event space). These notifications are distributed by the **distributor components** to many **consumer components** (log systems or final users) that monitor these notifications. A distributor or a probe may process the information before it is delivered. At this point, filters, aggregation and pattern matching techniques can be applied, in order to generate higher-level events. This approach reduces the traffic of data sent from the probes to the consumers, coping with scalability.
- **EventQueue.** The EDEM system monitors Java UI events generated by widgets, these events are stored in an event queue, that can be queried by the probes. Probes intercept these events from the windowing system (Java AWT or Swing) and distribute these events by e-mail or generate a log in a file. This data can be further, or immediately, processed by the usability team (consumers), which can perform changes in the UI project.
- **Agents.** Agents specify triggers, guards and ac-

tions. They are defined in a GUI editor. Agents are stored in ASCII format in text files. They can be stored locally or can be downloaded from a HTTP server. They "configure" the Probes with the adequate user expectations. Agents can be configured to monitor notifications from other agents. This facility can be used to compose agents hierarchically to detect patterns of events at increasing levels of abstraction. Agents can be added or removed at runtime, since the probes are not compiled with the application.

The EDEM system also allows the specification of **actions** that can be performed in response to observations, processing and notifications. These actions can evolve the execution of specific programs, collecting user feedback, manipulate probes and so on.

6.4. Integrating EDEM With the Application

EDEM is implemented in Java and is used to monitor UI events in applications implemented with this language. The integration of EDEM with the application requires only a slightly modification in the application source code:

```
import edem.api.*;

// Main class of the program
public static void main(…) {

// initialize EDEM
    Edem.begin();

    (…)
}

// During the end of the program
public finalize() {

// exit EDEM
   Edem.end();

}
```

The *Eden.begin*() operation starts the monitoring of the application while the *Eden.end*() finishes this operation.

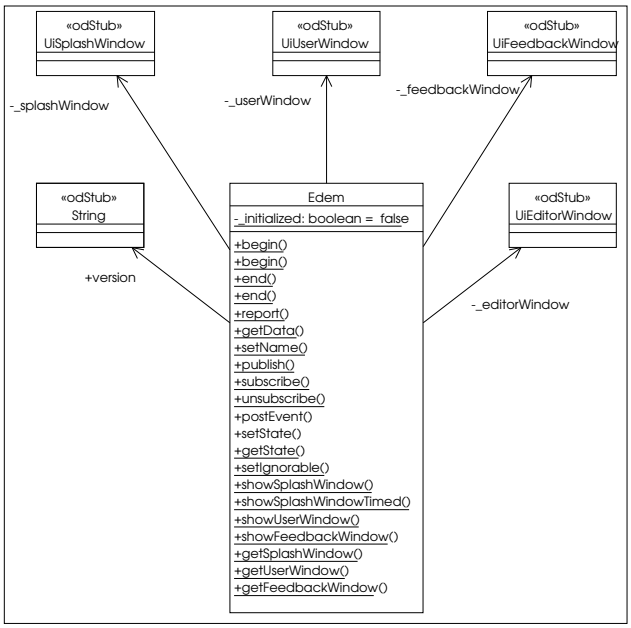

Figure 5 EDEM API Class Diagram

Elements to be monitored can be given a specific name using the *setName*() operation in the EDEM API, for example, a JTextField called myTextFiedl:

```
Edem.setName(myTextField, "Any Name");
```

6.5. Current Status

According to the author, the system is still being developed. In this process, some challenges need to be addressed in many areas as: agent representation and maintenance; data storage and analysis; integration of expectations to the development process; user privacy and non-disruptive techniques for collecting user feedback.

EDEM have some similarities with fields like application usage monitoring, software process monitoring and distributed system monitoring and debugging. In these fields, (distributed) data collection is useful for the understanding, evaluation and maintenance of these systems.

7. Integrating EDEM with Lumière Architecture

The expectation agents defined in EDEM allow the capture and log of sequence of events defined by the trigger of these expectation agents. For performance reasons, this information is stored in a local log file, which is sent to the EDEM server at the end of a monitoring session. It was not designed with the intent of monitor a very fine granularity of events, but only those sequence of events

that violate a user expectation. The EDEM Event queue is an internal component of the system and is not visible outside the application.

This approach is incompatible with the periodic check of events proposed by the Lumière Architecture (see section 4.3). Hence, The EDEM system need to be adapted to provide:

- Collect fine grain event, as they are generated by the application, in order to support to the development of the Lumière Event API, and to allow the composition of low-level events with higher level observations defined in this language.
- Add timestamps to the events in order to support the implementation of the Lumière Event API operations.

7.1. Adapting the EDEM System

Our approach aims at making the least modification to the kernel of the EDEM system. According to this principle, we propose the following adaptation:

7.1.1. Capturing Low-level Events

In order to cope with the fine-grained event collection approach of Lumière, without accessing the EDEN event queue, we propose de definition of one agent for each element of the interface that need to be monitored. For example, if monitoring of any event related to a ResetButton in an application. needs to be monitored, an agent could be defined as follows.

```
version="EDEM Version 2.0", email="EDEM
Administrator <user@company.com>",
comments="Application Name",
agents=Vector[1,
edem.kernel.Agent[name="Agent 1",
comments="",
beginTrigger=edem.kernel.Trigger[operator=0
, elements=Vector[1,
edem.kernel.TriggerRecord[eventSpec="ANY_EV
ENT|Window/System/ResetButton|null"]3]
]2,
dataTrigger=edem.kernel.Trigger[operator=0,
elements=Vector[0]
]4,
endTrigger=edem.kernel.Trigger[operator=0,
elements=Vector[0]
]5,
beginTriggerGuard=edem.kernel.TriggerGuard[
operator=0, elements=Vector[0]
]6,
endTriggerGuard=edem.kernel.TriggerGuard[op
erator=0, elements=Vector[0]
]7,
beginGuard=edem.kernel.Guard[operator=0,
elements=Vector[0]
]8, endGuard=edem.kernel.Guard[operator=0,
elements=Vector[0]
]9,
beginAction=edem.kernel.Action[eventSource=
0, eventName="", elements=Vector[0]
]10,
endAction=edem.kernel.Action[eventSource=0,
eventName="", elements=Vector[0]
]11,
eventData=edem.kernel.EventData[dataType=0]
12,
stateData=edem.kernel.StateData[dataType=0,
elements=Vector[0]
]13,
userData=edem.kernel.UserData[dataType=0,
dataHeader="", dataMessage=""]14]1]
```

Another important point is the creation of a mechanism that allows the interface events to be captured outside the EDEM system.

The EDEM allows the specification of generic actions to be performed when an event is generated. This facility is provided by the following adaptation points in the class */edem/kernel/Agent.java*

```
//---------------------------------------
// action methods
//---------------------------------------
public void performBeginAction(EdemEvent e)
{
  // subclasses should override this method
  // to perform custom actions...
       eventCollector.setEvent(…);
}
public void performEndAction(EdemEvent e) {
  // subclasses should override this method
  // to perform custom actions...
}
```

We propose the addition of operations such as the one in bold face in the code above, to send information about the current event, to an external *EventCollectror* component.

The extension of these methods can be done by the modification of the EDEM code or by extending this class in another sublcass. The last approach is more generic, but requires the creation of an *AbstractFactory* design pattern [Grand98] that creates the instances of specific classes based on configuration parameters.

Mechanisms that allow the specification of the *EventCollector* are also necessary. The EDEM approach is to pass this information through command line or environment variables. A specific operation in the EDEM API can also be created to this specific propose. We propose the addition of the *configureEventCollector*() operation in the EDEM API. These extensions are presented in Figure 6.

7.1.2. Getting Time Stamps

One approach to this problem could be the extension of the */edem/kernel/EdemEvent.java* class to include the current timestamp. This would require the modification of the kernel elements of the EDEM system. A better approach in our opinion, is the timestamping of the events as they are generated and collected.

The timestamp is captured by the *CollectorAgent*, our specialization of the *Agent* class, and appended to the event associated to the agent.

7.1.3. Creating a Separate Queue

A separate *EventQueue*, for the specific propose of the Event Analysis must be created. This queue is populated with the continuous data sent by the detection of the agents population of agents defined to monitor the target application. This approach allows the manipulation of the queue events in an independent way from the internal queue of the EDEM system, allowing the implementation of proprietary optimizations for the sake of implementing the Lumière Event API.

A schema of this approach is described in the Figure 6 below. The *CollectorAgent*, an specialization of the *edem.kernel.Agent* class invoke *newEvent*() in the *eventQueue* component.

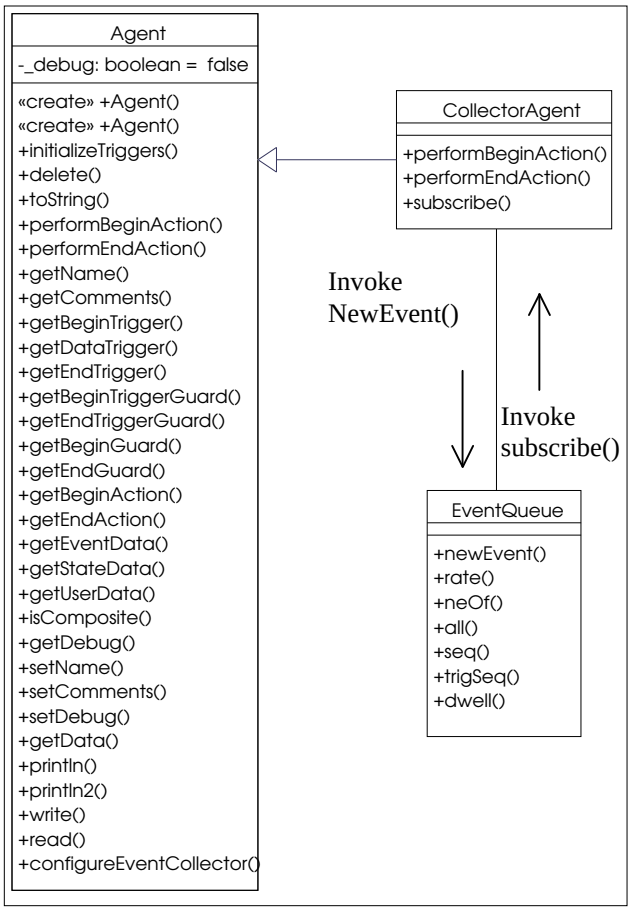

Figure 6 Proposed EDEM Extension

Once the events are collected and stored in the EventQueue, the implementation of the Lumière API is not difficult. The API calls are also presented in Figure 6.

7.2. Another Approach

A more generic approach to this problem is proposed by Cleidson R. B. de Souza [Souza01] in his paper for this lecture. In his approach, the EDEM was modified to provide a Publisher-Subscriber [Grand98] API. A publisher, the EDEM system, provides the events constantly to all the subscribers of the event of the application.

In order to populate our new event queue, a *ConcreteSubscriber* is implemented. This subscriber registers itself with the Publisher EDEM API implemented by Cleidson and start receiving notification of events.

This approach does not modify the EDEM events. In order to provide the necessary timestamps to our implementation, this data must be appended to the events as soon as they are collected in the EDEM.

An *EventCollector* that extends the *ConcreteSuscriber* have to be implemented in order to gather the events in a

timestamped queue. Cleidson's approach is presented in Figure 7.

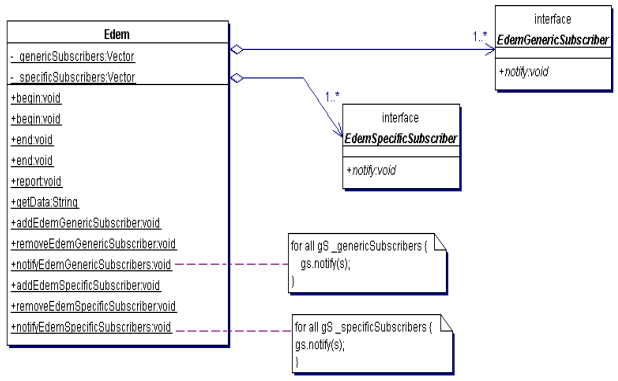

Figure 7 A Publish-Subscribe model for EDEM Event Collection

8. Discussion and Future Work

We proposed in this paper and extension to the EDEN monitor system, that allows this usability collector application to be integrated with a knowledge-based system. In special, we propose the integration of EDEN with an active help system. We use the Lumière project as the basis of this architecture, and a source of requirements for our project.

Further implementation and testing of this approach is also necessary. The evolution of this project to a full-fledged knowledge-based system is not a trivial task. However, we can envision a system that integrates the support provided by the EDEM system in the collection of events, with the knowledge representation capability of the Bayesian networks. In special, the free version of this system provided by Microsoft could be used to realize this big picture.

A big gap exists between the design and the implementation of these systems. More challenges also arise in the compression of the techniques used in the implementation of a user model using Bayesian Networks. The implementation and update of user profiles, the determination of thresholds for each user to avoid intrusiveness, an also issues in this implementation.

9. References

[Fischer00] Fischer G. User Modeling in Human-Computer Interaction. Contribution to the 10th Anniversary Issue of the Journal "User Modeling and User-Adapted Interaction (UMUAI)" (in press) [http://www.cs.colorado.edu/ ~gerhard/papers/umuai2000.pdf]

[Fischer99] Fischer G. User Modeling: The Long and Winding Road, Proceedings of UM99 (User Modelling) Conference (Editor: Judy Kay), Banff, Canada, Springer Verlag Wien New York, pp. 349-355. June 1999. [http://www.cs.colorado.edu/ ~gerhard/papers/um99.pdf]

[Grand 98] Grand, Mark. Patterns in Java: a catalog of reusable design patterns illustrated with UML, Wiley Computer publishing: John Wiley & Sons. Inc, 1998.

[HBHHR98] Horvitz E., Breese J., Heckerman D., Hovel D., and Rommelse K.. The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, July 1998.

[Helberg98] S. R. Helberg. Is IA Going Mainstream At Least? A Look Inside Microsoft Research. IEEE Intelligent Systems (March, April '98). 1998. [ftp://ftp.research.microsoft.com/pub/ejh/x2xins.lo.pdf]

[HGC95] Heckerman, D., Geiger, D., and Chickering, D. Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning, 20(3), pp. 197-243. 1995.

[HH98] Heckerman D. and Horvitz E.. Inferring Informational Goals from Free-Text Queries: A Bayesian Approach, Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI. July 1998. http://research.microsoft.com/ ~horvitz/aw.htm.

[Hilbert99] Hilbert D. Large-Scale Collection of Applicatoin Usage Data and User Feedback to Inform Interactive Software Development. Technical Report. Department of Information and Computer Science, Irvine, UCI-ICS-99-42. October 1999.

[Horvitz99] E. Horvitz. Principles of Mixed-Initiative User Interfaces.Proceedings of CHI '99, ACM SIGCHI Conference on Human Factors in Computing Systems, Pittsburgh, PA, May 1999.

[HR98a] Hilbert D. M. and Redmiles D. F.. Agents for Collecting Application Usage Data Over the Internet. Proceedings of the Second International Conference on Autonomous Agents (Agents'98). 1998. (Originally Technical Report UCI-ICS-97-41).

[HR98b] Hilbert D. M. and Redmiles D. F.. An Approach to Large-Scale Collection of Application Usage Data Over the Internet. Proceedings of the 20th International Conference on Software Engineering (ICSE'98), 1998. (Originally Technical Report UCI-ICS-97-40).

[HR98c] Hilbert D. M. and Redmiles D. F.. Separating the Wheat from the Chaff in Internet-Mediated User Feedback. Proceedings of the Workshop on Internet-based Groupware for User Participation in Product Development held at CSCW'98, 1998.

[LJS99] Linton, F., Joy D., and Schafer H.. Building User and Expert Models by Longterm observation of application usage. In: Proc. 7th Int. Conf. on User Modeling. Banff, Canada, pp. 129-138. 1999.

[MSBN] Microsoft Belief Network. http://research.microsoft.com/msbn/default.htm

[RR00] J. E. Robbins and D. F. Redmiles. Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML. Information and Software Technology 42, (2000), 79-89

[Souza01] C. R. B. De Souza. Providing Awareness with EDEM and Cassius. ICS208 Final Paper. Department of Information and Computer Science, University of California, Irvine. Winter 2001.

[XBNFF] XML Belief Network File Format: Main Page. http://research.microsoft.com/dtas/bnformat/

Kobsa A.. Generic User Modeling Systems. To appear in User Modeling and User-Adapted Interaction, Tenth Year Anniversary Issue. 2000. http://www.ics.uci.edu/ ~kobsa/papers/2001-UMUAI-kobsa.pdf

Kobsa, A. User Modeling: Recent Work, Prospects and Hazards. In: M. Schneider-Hufschmidt, T. Kühme and U. Malinowski, eds. (1993):Adaptive User Interfaces: Principles and Practise. Amsterdam: North Holland Elsevier. 1993. [http://www.ics.uci.edu/ ~kobsa/papers/1993-aui-kobsa.pdf]

Pohl, W., A. Kobsa and O. Kutter . User Model Acquisition Heuristics Based on Dialogue Acts. International Workshop on the Design of Cooperative Systems, Antibes-Juan-les-Pins, France, 471-486. 1995.