MAITH: A Meta-software Agent for Issue Tracking Help

Touby Drew
Medtronic, Inc.
7000 Central Avenue NE; MS RCE290
Minneapolis, MN, 55432-3576
touby.drew@medtronic.com

Maria Gini
University of Minnesota
4-192 EE/CSci, 200 Union St SE
Minneapolis, MN 55455
gini@cs.umn.edu

ABSTRACT

Issue tracking is an essential part of regulated software development where it is typically supported by software systems which are complex and not easily customizable. We propose a meta-software agent that senses what windows and widgets are in focus by the user and leverages this awareness to provide support. The user is given ways of making and recalling annotations appropriate for the context. By observing users in action the agent creates models which can then be used to predict and suggest next steps. This paper describes an early prototype of this approach built as a proof of concept. Preliminary results and directions for future work are outlined.

Categories and Subject Descriptors

I.2,11 [**Distributed Artificial Intelligence**]: Intelligent Agents

General Terms

Design, Algorithms, Human Factors, Architecture

Keywords

Meta-software Agent, Regulated Software Engineering, Issue Tracking Support, Applications

1. INTRODUCTION

Issue tracking is an important aspect of large scale, regulated, software engineering domains and is typically supported by a complex system of programs and users which work together to document and manage change of process, products and other essential artifacts.

This paper focuses primarily on a novel approach to extending an issue tracking system (ITS) implementation used by hundreds of employees in a division of the world's largest implantable medical device manufacturer. Important constraints of this domain include (1) restrictions on changing the existing qualified system, (2) the need to support and follow users as they interact with the system through different programs and technologies, (3) the need to support users without having prior knowledge of their goals or requiring a predictive action-sequence model of their current task.

Cite as: MAITH: a Meta-software Agent for Issue Tracking Help, T. Drew and M. Gini, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 1755-1762

Within these constraints we propose a Meta-software Agent for Issue Tracking Help (MAITH) which interacts with the existing issue tracking systems as meta-software, performs feature-based recognition of the current virtual context, and provides context relevant user support. MAITH has been created as a proof of concept. It currently exists as a prototype for exposing various technical, architectural, and usability challenges as well as ultimately taking the first steps in applying a meta-software agent approach to supporting work within Medtronic Neuromodulation as described in this paper. Based on the work completed so far, MAITH has good potential to be tested with a small group of workers and, depending on its success and the degree of interest, refined and deployed to a much larger number of users.

The manner in which MAITH supports its complex application domain is different from previous approaches to integrating agents with user interfaces (UI) (see Section 6). MAITH reduces significantly the requirements on the target applications because it determines the context using two features, i.e. the window and widget currently in focus, which are provided by a layer of software below the applications themselves. MAITH supports polymorphic and configurable context modeling, which enables generalization by changing the type of context descriptor, and addition of new types of descriptors. MAITH also supports a multimodal UI, which allows users multiple ways to interact with the agent, such as using it for unanticipated advice, quick control, or intense use. Finally, MAITH provides support in different types of applications, such as those where sequential information is a reliable predictor of the next user context as well as applications where the sequence varies widely but the current state alone is a good predictor of appropriate help.

More specifically, key contributions of this work include:

1. proof of concept design of a meta-software agent for issue tracking systems
2. a novel suggestion that widget and window in focus are the two key features for meta-context observation and user support;
3. a prototype implementation with
   (a) support for monitoring and transitioning between active elements in Web, application, file system, Eclipse, and other software;
   (b) constant user observation with reasonable performance;
   (c) a sophisticated multi-modal UI for direct two-way user interaction, notification of context rel-

evant information, and configuration and interaction with model and data;

(d) context-sensitive annotations by users using context descriptors that enable polymorphic context matching;

(e) a method for context prediction that uses the window and widget in focus.

4. a unique approach to supporting issue tracking in a regulated engineering domain.

In the remainder of this paper, we present (1) background on the regulated issue tracking domain and motivation for supporting it, (2) a conceptual introduction to MAITH, (3) the architecture and design of MAITH, (4) examples of expected usage and results, (5) discussion including relation to existing work, and finally (6) conclusions from our preliminary experience and potential future directions.

2. APPLICATION DOMAIN BACKGROUND

Teams working to develop and maintain software and other artifacts relevant to implantable medical devices work with numerous and changing tools, processes and artifacts that are evolved and maintained to support sophisticated and critical systems. These teams and systems operate and evolve over decades of development under competitive, regulatory, and other business pressures. As a result, the efforts required to master and maintain them by dynamic teams of engineers are daunting and expensive [19, 8].

The Production Neuromodulation Issue Tracking System (PNITS) currently used by Medtronic Neuromodulation, Inc. is no exception. PNITS is based on a heavily customized implementation of the well known IBM Rational ClearQuest [1] system. The system is accessed primarily through Web, Eclipse, and native Windows-based clients for a variety of purposes. These include capturing, communicating, and adding clarity to the process being followed and the status of key aspects of the work such as review and work completion. Issues are typically related to projects, users, other issues, classifications and other tools (e.g. files cannot be promoted into the formal integration/build environment unless they are associated with an issue which in turn is associated with a relevant project and in the resolving state).

After initial peer reviews and before final closure review, a separate team of quality assurance engineers regularly review and reject issues, sometimes several times and long after the related work has been dormant. Despite various meetings, discussions, supporting plans, processes, work instructions and required trainings, metrics collected on one large team showed that its members repeatedly struggled with the issue tracking system. In the first several months of their work, more than half of the issues submitted and peer reviewed were later rejected, many of them multiple times, often by quality engineers. After more than a year of experience and several changes to how the issue tracking system is used by that team, the rejection rate has improved but not stabilized below 20%. Users, including the software development function's leadership, have discussed various changes to the system itself, but, due to the number and range of users, effort and complexity involved with making and especially qualifying any changes, and the need for tightly regulated customization and extension of the software, this is not a viable option.

3. A LESS DIRECT APPROACH

Domain constraints limit the applicability of more direct approaches such as altering, extending, or making the PNITS more customizable. Adding to the complexity is the fact that several client programs and technologies have to be supported. The need for an alternative approach seems apparent.

Much of today's virtual content is ultimately exposed to human users as part of a dynamic collection of separate windows and widgets, a subset of which may be in focus and receive input or provide output at any given time. This control of focus and information exchange is primarily driven and understood by the user and, to a lesser extent, by the logic of a particular software program associated with the windows and widgets in question. In some cases, however, "meta-software" monitors and interacts with the virtual context, determining (and/or controlling) what windows and widgets have focus and interacting with them through content retrieval, mouse and keyboard input. Such meta-software is often used to carry out scripted or more complex UI automation for software verification and validation purposes. In some cases UI behavior can be automatically recorded, edited, and played back to facilitate verification and validation activities or within a particular program by an end user for other reasons.

We propose a meta-software approach to provide support for PNITS. We want the software to observe the PNITS clients, much as a user would, while the user interacts with them. Unlike typical meta-software we want to be able to watch the user's context, have some recognition of what is going on with the issue tracking client, and interact with the user as an "advisory agent" [14] while refraining from taking over the user's role in the issue tracking system. Unlike macros, verification tools and other nascent meta-software we want to be able to monitor and recognize the context across and inside the diverse Windows-native, Web, and Eclipse client technologies.

Given the complex, varied, and often short or unique interactions that users have with PNITS, the meta-software should be so loosely coupled as to allow and even encourage the user to transition painlessly in and out of related and unrelated tasks and programs. This suggests that the Meta-software should act as an Agent for Issue Tracking Help (MAITH). It should be able to integrate different kinds of "sensed" inputs including the window and widget in the user's focus across programs and technologies and to derive "features" from that input, to recognize and react to different contexts described by those features, and so on.

4. BASICS OF MAITH DESIGN

MAITH is architected to monitor the current virtual environment through virtual sensors, resolve these inputs into features, use those to identify the current context, and finally work with its understanding of the current context and user input to manage and perform actions (see Figure 1). It can, for example, determine the window and widget currently in focus, record an annotation associated with this context, and recall the annotation at some point in the future when it recognizes the relevant context.

4.1 Monitoring

MAITH relies heavily on two key features which reflect (1) the currently active window and (2) the currently ac-

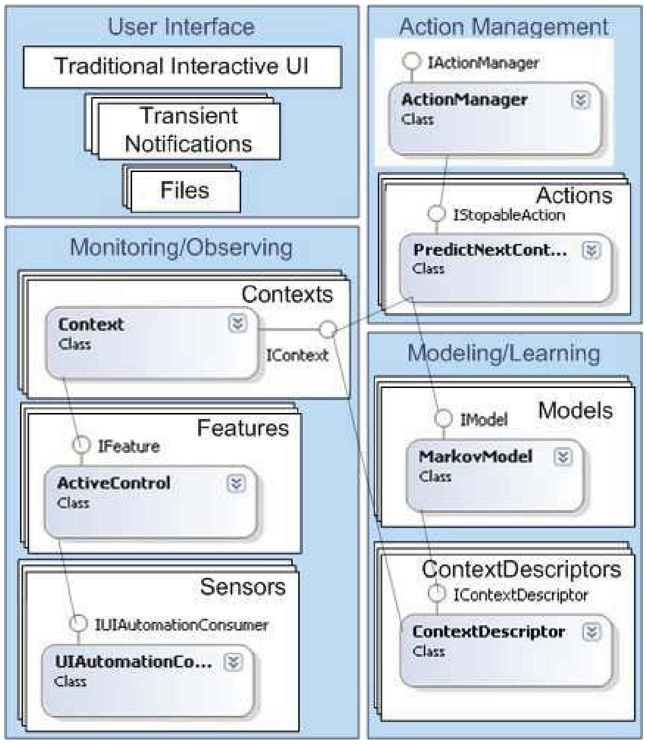


Figure 1: Architectural sketch with example classes and interfaces

tive widget. This avoids over-fitting, and remains performant while continuously operating in the background, and operating much as a user would. These features are supported primarily by custom virtual sensors based on the UI Automation framework [11]. These sensors are running in a separate process, which is essential to avoid threading issues [2]. This approach further supports recognition of when the web client is in use by observing the active window. In addition, MAITH needs to generalize this recognition to deal with different browsers. This is accomplished by using a more general ContextDescriptor than an exact match on the active window.

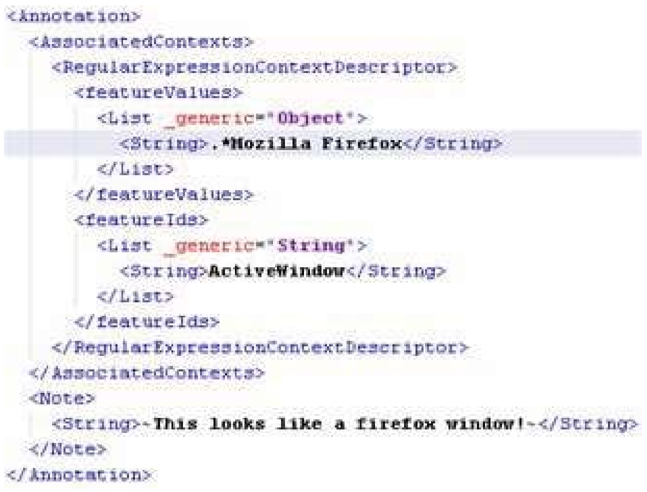
While sufficient to monitor activity within most technologies, the UI Automation based approach to monitoring the active widget is not able to function with the complex and customized web client. To address this, we developed a new Firefox plugin to monitor, and expose through a telnet interface provided by MozRepl [15], the active widget within a web application. A FirefoxWatcher was developed and integrated to supplement the UI Automation sensor input, allowing details of how the "active widget" is sensed to be encapsulated under a single feature.

4.2 User Interfaces

MAITH often observes users' effects on the virtual context and operates quietly in the background, but it also interacts directly with users primarily through three different interfaces:

1. through a traditional interactive window and tray icon which receive direct run-time user input,
2. through files for configuration and context related data management, and finally
3. through unidirectional user notifications of context specific information.

Users can control what MAITH is doing to support and consider their actions, how it is using its files, and review its status primarily by selecting actions from its interactive window (which they can bring up from its tray icon). Some users who need to tune or have greater control over the agent may interact with it by creating or editing files which it understands, for example as seen in Figure 2 to generalize when it should recognize (and potentially thus react to) a certain context by simply removing irrelevant features from its description and/or using more sophisticated syntax (in the toy example shown using a more sophisticated context descriptor type and leaving only the 'ActiveWindow' feature). Finally, MAITH notifies the user of context relevant information through transient messages that are shown and fade away just above its tray icon (see various figures in the sections below).



```
<Annotation>
  <AssociatedContexts>
    <RegularExpressionContextDescriptor>
      <featureValues>
        <List _generic="Object">
          <String>.*Mozilla Firefox</String>
        </List>
      </featureValues>
      <featureIds>
        <List _generic="String">
          <String>ActiveWindow</String>
        </List>
      </featureIds>
    </RegularExpressionContextDescriptor>
  </AssociatedContexts>
  <Note>
    <String>-This looks like a firefox window!-</String>
  </Note>
</Annotation>
```

Figure 2: Example of a generalized annotation

4.3 Modeling

As alluded to previously, MAITH is able to describe the current context and/or recognize when a certain context descriptor matches the current context using classes which meet an IContextDescriptor interface. The primary and simplest of these is ContextDescriptor, which maintains a set of value pairs each containing a feature identifier and a description of that feature's value. MAITH is able to create models by creating, retaining, organizing, and associating context descriptors with themselves and other content based on its observations and user input.

MAITH can associate context descriptors with user input to form annotations. MAITH allows collections of these annotations to be saved to files, loaded, manipulated, used to automatically display the relevant content whenever the agent observes that the current context matches the context described in one or more of a set of annotations. For example, MAITH allows for a user to provide it with content and demonstrate the context with which it should be associated to form an annotation, save the annotation, later load that and other annotations and have their content automatically displayed whenever relevant contexts are encountered by the user.

MAITH is also capable of associating context descriptors with other context descriptors and other information such as temporal directionality to form sequential context description chains. For example, MAITH can be asked to train a model on the set(s) of contexts it observes over one or more periods of time. Along these lines when a user asks MAITH to start predicting future contexts (which is described later in the paper) a running model (basically a circular buffer) of recent context descriptions is kept and used with each context change to search a previously trained, depth-limited, probabilistic, trie-variant-based, Markov-chain model to determine and display the most probable next context description. Typically past work in this sort of prediction has focused on action, language, or audio-visual patterns. The modeling we use is unique but draws heavily from previous work in these other areas. The relevant training and prediction algorithms are similar to algorithms that have been used for natural language processing and compression, and our implementation is similar in many regards to the implementation of Prediction by Partial Match 'Method C' (PPM-C) described by Begleiter [3]. [1]

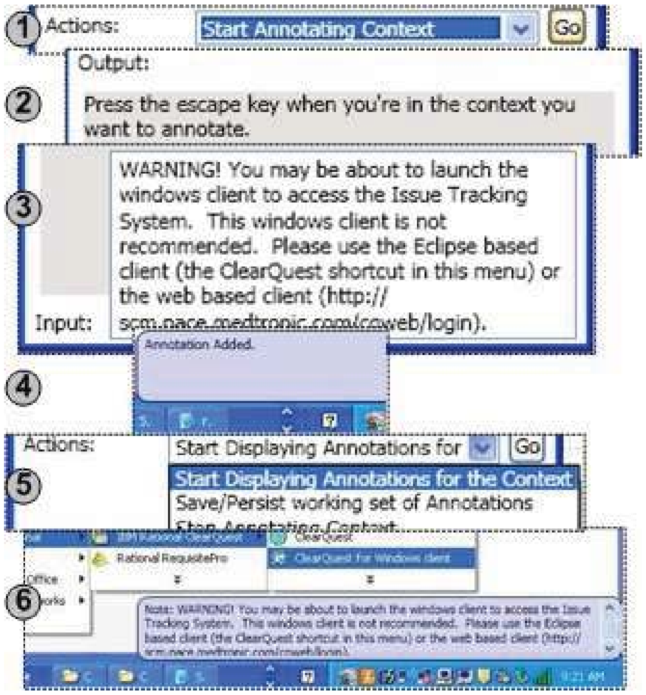
5. MAITH IN ACTION

The sections below describe four different examples of MAITH in action supporting PNITS.

5.1 The Right Client

With MAITH running on a user's computer and monitoring which window[2] is active, it determines when one of the Web, Eclipse, or (native-)Windows PNITS clients are being used and provides client-appropriate support.

MAITH should discourage users from using the Windows client because although it appears to function reasonably well, some idiosyncrasies (like not scrolling on one screen when it should) have lead to mistakes and significant rework. To train MAITH to prevent users from using this client, one user selects the Windows client on her machine and then asks MAITH to make a context appropriate note that this program should not be used and that one of the other portals should be used instead. MAITH then saves and the user generalizes the context-associated-annotation (NOTE) so that it should be displayed anytime a window with that title is opened. This and other annotations saved may be loaded automatically or manually, merged and displayed in later use. Figure 3 shows a sequence from an example of the process above.


1
Actions:
Start Annotating Context
Go
2
Output:
3
Input:
4
5
Actions:
Start Displaying Annotations for
Go
Start Displaying Annotations for the Context
Save/Persist working set of Annotations
Stop Annotating Context
6
Rational RequisitePro
ClearQuest
ClearQuest for Windows client
Office

Press the escape key when you're in the context you want to annotate.

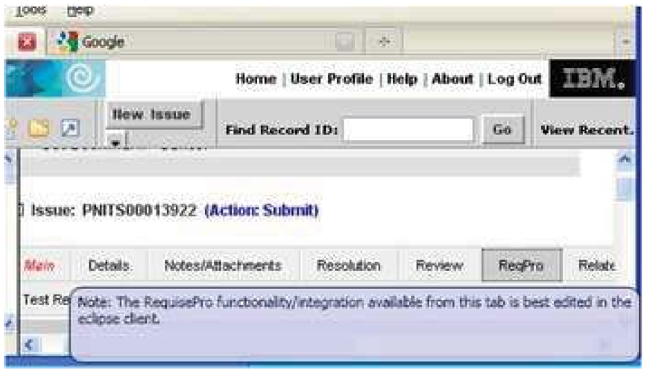
WARNING! You may be about to launch the windows client to access the Issue Tracking System. This windows client is not recommended. Please use the Eclipse based client (the ClearQuest shortcut in this menu) or the web based client (http://scm.pace.medtronic.com/cqweb/login).

Annotation Added.

Note: WARNING! You may be about to launch the windows client to access the Issue Tracking System. This windows client is not recommended. Please use the Eclipse based client (the ClearQuest shortcut in this menu) or the web based client (http://scm.pace.medtronic.com/cqweb/login).

Figure 3: Example of a Windows client warning annotation sequence

The Web client and the Eclipse client each have different advantages and disadvantages and should be recognized by an agent monitoring a user's virtual environment in much the same way the Windows client is recognized. The Web client, being browser based, has the complication of not always having the same window (e.g. it could have been opened in Firefox as opposed to Internet Explorer), but this can be readily generalized using more flexible matching rules. When the Web or Eclipse client is recognized, MAITH can automatically or upon request offer the user additional support actions that are appropriate to the context, such as information resources, annotation, and identification of the widget which is typically accessed after the currently focused one.For example, if an issue is too large to be opened in the web client or the user goes to modify the ReqPro tab, which is only partially supported in the web client, MAITH refers the user to the Eclipse client (see Figure 4).



Issue: PNITS00013922 (Action: Submit)

Note: The RequisePro functionality/integration available from this tab is best edited in the eclipse client.

Figure 4: User being referred from web client to Eclipse client

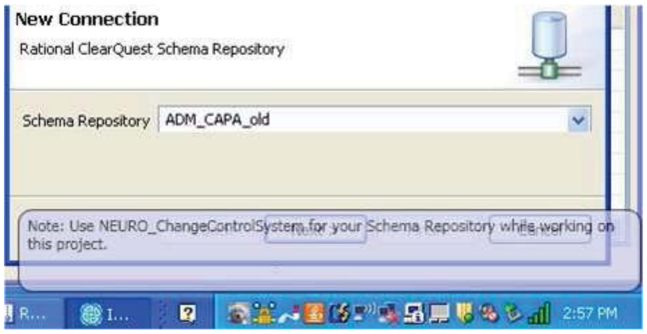
Though MAITH clearly must support each of the PNITS clients, for simplicity we focus the remaining examples on the Eclipse client.

[1]Differences include total count caching, option for simultaneous use during training, application to contexts descriptors rather than symbols, and exception for certain training conditions.

[2]Here the term 'window' is used loosely. While MAITH does indeed recognize the windows-native or Eclipse client by their window, this does not make sense for the web client. For example, we certainly would not want to recognize any Mozilla Firefox window as the web client, but would want to recognize when the active tab of Mozilla Firefox has loaded the web client. Furthermore, it is best to also recognize shortcuts to the Windows client from the "Start" menu to provide a relevant warning even before the program is launched.

5.2 Client Interface Annotation

In addition to helping users work with an appropriate client, the more detailed context within the client can be annotated and later recognized and supported. In this fashion, one or more users can create, modify, and share a set of groupware annotations. For example, before using the PNITS system, new users must all follow the appropriate process for establishing a connection to the database. MAITH can be used to guide users through this process by presenting them with context appropriate instructions and comments, which MAITH learned from watching and being instructed by previous users (see Figure 5).

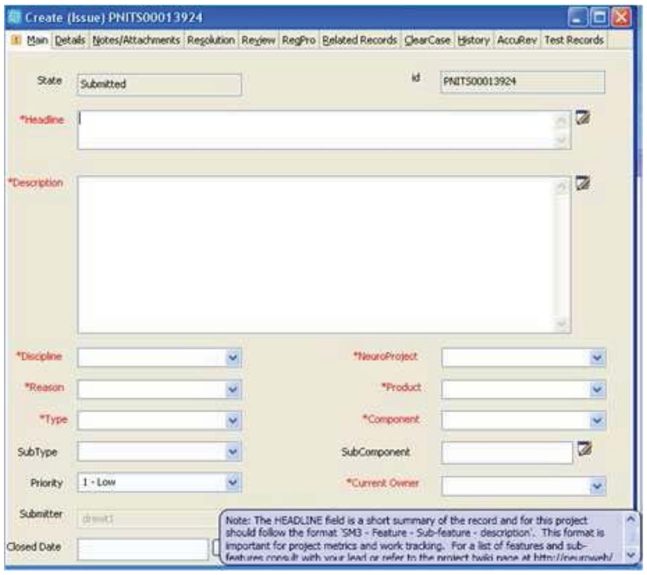

New Connection

Rational ClearQuest Schema Repository

Schema Repository: ADM_CAPA_old

Note: Use NEURO_ChangeControlSystem for your Schema Repository while working on this project.

Figure 5: Example of support in establishing new connections


Create (Issue) PNITS00013924

State: Submitted

id: PNITS00013924

*Headline

*Description

*Discipline

*NeuroProject

*Reason

*Product

*Type

*Component

SubType

SubComponent

Priority: 1 - Low

*Current Owner

Submitter: dreat1

Closed Date

Note: The HEADLINE field is a short summary of the record and for this project should follow the format 'SM3 - Feature - Sub-feature - description'. This format is important for project metrics and work tracking. For a list of features and sub-features consult with your lead or refer to the project twiki page at http://neuroweb/

Figure 6: Example group-specific support for a field

Along these lines, the way a certain group uses PNITS can be encoded and shared just with that team in a similar fashion. This is particularly important because of the degree of complexity of the system and its relationship with other systems used by that group. Consider as an example a large team of software engineers with somewhat regular turn-over but established rules for using PNITS. More specifically, as shown in Figure 6, although there are numerous "free format" text fields exposed by the system, the team has created specific rules about the format and content of several of these including the "Headline" field which is used in various project metrics and work tracking queries.

5.3 Sequences: Suggesting Next Steps

As previously discussed, a software agent can save a description of the current context and associate that with some other item, such as annotation to display, or more generally some action that could be performed in that context. Structures of context descriptions themselves may also be worth loading, saving, and organizing for use without associated actions. For example, sequential chains or traces of context information are a natural way to organize such context descriptions.

In the previous examples and in its anticipated use, MAITH training, context recognition, and support is performed without direct consideration of the immediate historic sequence of the virtual context. This allows MAITH to avoid "the inability of the ... system to track users when they performed sub-goals, such as navigation, using action sequences other than those within the script" [4] which would be common in this domain. However, in certain circumstances, this sort of sequential information is desirable, and MAITH can be asked to model observed context sequences and/or predict/suggest next steps having observed the recent history or even just the current context.

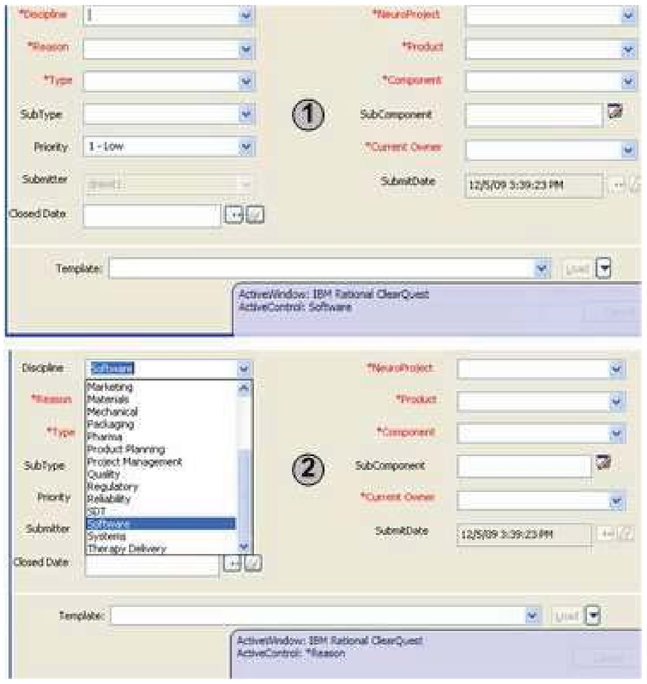
Although aimed at more physical contexts, Rahlff et al. describe the use of logging context timelines in Context Trace Technology (CTT) to provide a means for context matching/clustering within a body of context information to find "nearby" contexts which might provide personal (e.g. "when was I here last?") or possibly even social (e.g. "what do people usually do here?") context-aware support [17].

To explore a more sophisticated structure of organizing context descriptions (using counting trie based Markov chain structures) we examined the feasibility of extending MAITH to support learning the probability of context transitions by observing user behavior and predicting future contexts based on available current context timeline. This subjectively proved successful in predicting the next context (e.g. active window and focused widget) while remaining robust to variable length (or no) matching history.

To understand how this is applied in supporting the PNITS system, consider a scenario in which MAITH has previously observed and modeled the creation of new issues and is currently observing a user attempting to create a new issue. The new user could ask MAITH to begin suggesting the next action based on the model learned from its previous experience. The user could then attempt to complete the issue and look to MAITH for guidance with the assumption that the model in use may reflect sequences similar to the ones the user should now produce. Perhaps, more specifically the user needs to complete the drop down fields on the current tab but doesn't know where to start, how they are sequenced (they are interdependent), or what options to consider in them. As seen in Figure 7, MAITH can be helpful in cases like this; here first suggesting what to select and then where to go next by describing the next screen and widget in a note to the user.

5.4 Issue Tracking System Files

PNITS supports export and import of various query and data files. When a large number of these files are present they can be difficult to find, their content and use can be



Priority: 1 - Low

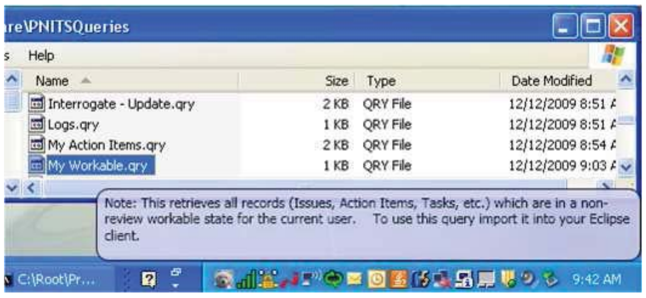
SubmitDate: 12/5/09 3:39:23 PM

Discipline: Software

ActiveWindow: IBM Rational ClearQuest
ActiveControl: Software

ActiveWindow: IBM Rational ClearQuest
ActiveControl: *Reason

Figure 7: MAITH using Markov Model to suggest selection (in 1) and next field (in 2)



Name	Size	Type	Date Modified
Interrogate - Update.qry	2 KB	QRY File	12/12/2009 8:51 A
Logs.qry	1 KB	QRY File	12/12/2009 8:51 A
My Action Items.qry	2 KB	QRY File	12/12/2009 8:54 A
My Workable.qry	1 KB	QRY File	12/12/2009 9:03 A

Note: This retrieves all records (Issues, Action Items, Tasks, etc.) which are in a non-review workable state for the current user. To use this query import it into your Eclipse client.

Figure 8: MAITH displaying an annotation of the selected query file

difficult to remember and slow to explore. Though traditional approaches to documenting such things exist, such as additional readme files, emails, properties of individual files, descriptive naming, and organizing folders, there is an opportunity for further improvement. MAITH allows these files and the folder structures that contain them to be annotated (see Figure 8) and otherwise supported [3]. Enabling one user to add a context specific annotation associated to a certain file or folder on their computer and have it generalized and shared with other users is desirable. This sort of annotation of items within a Windows Explorer window is in many ways similar to the annotation of a web or program interface described in the previous sections. As described previously MAITH associates such file system annotations with context descriptors that recognize the context regardless of how the window was opened and can easily be generalized. For example if deploying a set of shared files MAITH can recognize when any file (regardless of path) of certain name is selected. Or, for example, it can recognize all files within a certain folder or with a certain file ending and describe that they are queries and how they can be used.

6. RELATED WORK AND DISCUSSION

Most of the research in context-awareness "has focused on building infrastructures to support programmers in building applications and on the applications themselves despite the tremendous value this has in empowering end-users to build applications" [6]. Some more recent work (for instance, [6, 7, 9, 18]) embodies an emerging trend of allowing end-users to control how they leverage context-aware support. Differences in our approach involve (1) meta-software agents, (2) support for user involvement in generalization, and (3) sophisticated support for action control. Our approach is ultimately consistent with and further supports these goals.

Considering the MAITH prototype and its preliminary use (see Section 5), it is clear that a meta-software agent approach to supporting issue tracking systems is promising and merits further work (See Section 7). Furthermore, there are some advantages to this over alternate approaches, many of which are currently being employed without sufficient success (see Table 1).

The manner in which MAITH is designed to be meta-software in that it is only loosely coupled and primarily independent of the applications it supports, attempting to use them as a user might, may be important in its future extension and application.

This includes attempting to use a small number of features which are readily visible to and consistent with use of end-users. While traditional "UI events... (e.g. mouse movements...[and] keyboard presses)...have long been regarded as a potentially fruitful source of information regarding application usage" they are "are typically [so] extremely voluminous and rich in detail" that it is difficult to obtain "a level of abstraction that is useful" [12]. Our selection of "window focus events from the UI layer" from the sea of more traditional UI, temporal, file system, and other potential inputs is consistent with some recent work in context-based (rather than content based) information retrieval [10] where widget focus has also been considered but is confounded by caching.

Our approach also includes leveraging access to monitor these features and interact with users in a manner that is not only isolated from the rest of the MAITH architecture, but common across and outside of what is provided by the applications themselves. In contrast to previous cross-application user-support work [7], this means keeping special add-ons or plug-ins for target applications to a minimum.

In the context of Lieberman's conceptual framework for integrating agents with conventional applications, these are departures in the MAITH approach from traditional leverage of recordability (extensive, accessible events/commands) and/or examinability (exposed data structures, plugin interface or API). These differences reduce the challenges and limitations related to "cooperation from the target applications" and the need for parsing, dereferencing, balance of granularity in the event protocol, dynamic interface negotiation, and so on which limit generality and practicality [13].

Avoiding the need to manipulate widgets, MAITH nearly

[3]As an example, sequential support can be used to suggest the next nested folders to expand based on those most commonly used by the user or others in their group depending on how the model is trained.

Table 1: ITS support approaches

Approach	Status with ITS	Cons (Re: MAITH)
Meetings	In use. Required and team-specific.	Not in context[a], Maintenance[b]
1 on 1 support	In use as requested.	Cost, Availability
Documentation	In use. Plans, processes, twiki, etc.	Not in context, Maintenance
Change or Extend ITS	Rare. Discouraged except in most essential & common	Slow, Cost, Maintenance, Invasive[c]
Follow me documentation [4]	Not attempted.	Tech-specific[d], Sequential[e]
(In context) Annotation tools[f]	Not attempted.	Environment / Program Specific, No Sequential Support
Task environment for support [7]	Not attempted.	Maintenance, Provides limited support
MAITH	Proof of concept	Not applicable

[a] Not available at the specific times and places where needed

[b] Costly to update and recreate for specific groups.

[c] Requires changes to ITS or its use with significant ripples into re-qualification, documentation, training, and so on.

[d] Technology-specific and tightly coupled into the Eclipse environment (including events, performance specifics such as quiescence, and so on). Cannot support web client, etc.

[e] Heavily dependent on sequential information; could not support more general context annotation as is more generally desirable in the application. See Section 5.3

[f] Such as 'shared annotation' [5], file system, and application-focused systems.

eliminates the need for widget understanding and minimizes concerns about inappropriate automation and the need for qualification and retraining. MAITH's provision for complex action control, directed and passive learning, and basic user input capabilities further abstract it from the applications at hand. These aspects of our meta-software approach are part of what defines MAITH as an agent and allows it to be more robust to changes, avoid over-fitting, and operate within domain constraints.

7. FUTURE WORK

There is a wealth of future work which includes testing with users, group support, and multiple other extensions.

A key next step is putting MAITH in the hands of users. Practical use will allow for the value of MAITH to be measured in terms of user perceived value, reduction in mistakes and rejections, improved consistency, reduced overhead, and ultimately monetary value to regulated businesses and industries. Additionally, we hope that broader exposure will bring back vital feedback that will guide, measure, and magnify the value of this work.

One aspect of MAITH which has been neglected is its potential to be supporting multiple users within a group with each user having their own agent. The current ability to save annotations and other information into files that users can swap and combine allows users to use MAITH collectively. Recent work "suggests that expertise delivered by social annotation mechanisms is helpful to users in learning unfamiliar domains" when considering document/content annotation [16] and we believe that this will hold true for more general (e.g. context, interface, and application) social annotation as well. Addressing, considering, and potentially building into MAITH support for such group collaboration would be an improvement to the current limited, manual group support. One potential approach could be to extend multiple instances of MAITH to collaborate in a distributed fashion, perhaps leveraging multiagent systems technologies.

With the existing design for MAITH providing for extension where key interfaces are used (see Figure 1), there is opportunity to explore additional implementations of actions, sensors, features, models, and context descriptors. This could allow for automated review support, network availability as part of the context, and various other extensions.

The MAITH architecture also allows for improvements. For example, after broader user testing, we may find that while performance issues were not found with MAITH during our work, users require an architectural change to provide them with control over when the agent is sensing their activity (currently it is always monitoring, but not always modeling or acting). Perhaps the most important architectural extension would be to allow for plugins (e.g. behind any or all of the interfaces described previously), which could facilitate broader user development, allow for easier configuration, and add utility to MAITH as a potential platform for exploring and applying this sort of approach more broadly.

This work may also be extensible to other domains where one or more users work in complex, customized, or evolving virtual environments. As such, because this approach remains context aware, but generally independent from and extensible across the different programs, files and technologies that define this virtual context it could potentially be applied to allow users to collaborate or be guided through whatever combination of virtual resources (e.g. files, folders, etc) and applications which are relevant to their domain. A particularly interesting area of application might be in education where students might overlay their virtual environment with guidance, annotations or other support from themselves or others and might allow their teachers to monitor their time spent in and progression through the relevant environment. Teachers could even look at the quality of their contributions and their impact in supporting other students or the context in which specific questions are raised. Perhaps some aspects of the approach to MAITH could inspire work on a similar meta-software agent approach to ultra-general, virtual context sensitive prediction and/or annotation support. Or perhaps in user interface design/revision with annotation, tracking of use performance metrics, etc. Within the medical device domain it could be useful in retrospective support (e.g. added to help familiarize future or released systems engineers with the product or previously used process and artifacts).

One possible promising direction for additional work in this area includes not simply supporting binary context matching, but more subtle evaluation of proximity to match, possibly in terms of number of matching features or more sophisticated context description proximity calculation (which might involve supporting user involvement in defining how to evaluate proximity for example in weighting criteria, defining patterns, etc). Considering the possible parallels between historic context chains and relevant examples provided more traditionally by mentors underscores the potential value of Context Trace Technology [17] in helping not just in simple context-sensitive decision points (as seen in Section 5), but in more complex tasks such as free form issue description or even other software engineering tasks that engineers might struggle with. Previous work has shown that "studying from examples and abstracting them for application in a new context is a common way of learning in programming that has been observed extensively in both new and experienced programmers" [20] and, intuitively, automating the process of providing examples and the possibility of quantifying and optimizing their proximity or relevance to the task at hand may be of value.

8. CONCLUSIONS

Issue tracking, and other regulated software development, is an important and challenging domain which, based on our preliminary experience with MAITH, appears to naturally lend itself to a meta-software approach. MAITH has taken the first step in establishing the concept of meta-software agents as a viable and valuable approach to issue tracking support. With a multitude of potential future work, user testing appears to be a key next step in this direction. Ultimately, MAITH may have the potential to significantly improve the quality and cost of issue tracking thereby improving productivity, and perhaps even products, which translates into significant monetary savings. This approach and the technologies relevant to it merit further evaluation, development, and application.

9. REFERENCES

[1] IBM - rational ClearQuest - rational ClearQuest - software. http://www-01.ibm.com/software/awdtools/clearquest/.

[2] UI automation threading issues. http://msdn.microsoft.com/en-us/library/ms788709.aspx.

[3] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[4] L. Bergman, V. Castelli, T. Lau, and D. Oblinger. Docwizards: a system for authoring follow-me documentation wizards. In *Proc. ACM Symposium on User Interface Software and Technology*, pages 191–200, New York, NY, USA, 2005. ACM.

[5] A. J. B. Brush, D. Bargeron, J. Grudin, and A. Gupta. Notification for shared annotation of digital documents. In *Proc. CHI (Conf. on Human Factors in Computing Systems)*, pages 89–96, New York, NY, USA, 2002. ACM.

[6] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. a cappella: programming by demonstration of context-aware applications. In *Proc. CHI (Conf. on Human Factors in Computing Systems)*, pages 33–40, New York, NY, USA, 2004. ACM.

[7] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *Proc. 10th Int'l Conf. on Intelligent User Interfaces*, pages 75–82, New York, NY, USA, 2005. ACM.

[8] T. Drew and S. Goetz. Vision of the virtual programmer - steps towards change in instrument systems for implantable medical devices. In T. F. B. Filho and H. Gamboa, editors, *Proc. Int'l Conf. on Biomedical Electronics and Devices*, volume 1, pages 156–159, Funchal, Madeira, Portugal, January 2008. INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

[9] K. Gajos, H. Fox, and H. Shrobe. End user empowerment in human centered pervasive computing. In *Proc. 1st Int'l Conf. on Pervasive Computing (Short paper)*, pages 134–140, Zurich, August 2002.

[10] K. A. Gyllstrom, C. Soules, and A. Veitch. Confluence: enhancing contextual desktop search. In *Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 717–718, New York, NY, USA, 2007. ACM.

[11] R. Haverty. New accessibility model for Microsoft Windows and cross platform development. *SIGACCESS Access. Comput.*, 82:11–17, 2005.

[12] D. M. Hilbert and D. F. Redmiles. Extracting usability information from user interface events. *ACM Comput. Surv.*, 32(4):384–421, 2000.

[13] H. Lieberman. Integrating user interface agents with conventional applications. In *IUI '98: Proceedings of the 3rd international conference on Intelligent user interfaces*, pages 39–46, San Francisco, California, United States, 1998. ACM.

[14] H. Lieberman and T. Selker. Agents for the user interface. In J. Bradshaw, editor, *Handbook of Agent Technology.* MIT Press, 2003.

[15] M. Mirra. MozRepl | hyperstruct. http://hyperstruct.net/projects/mozrepl.

[16] L. Nelson, C. Held, P. Pirolli, L. Hong, D. Schiano, and E. H. Chi. With a little help from my friends: examining the impact of social annotations in sensemaking tasks. In *Proc. CHI (Conf. on Human Factors in Computing Systems)*, pages 1795–1798, New York, NY, USA, 2009. ACM.

[17] O.-W. Rahlff, R. Kenneth Rolfsen, and J. Herstad. Using personal traces in context space: Towards context trace technology. *Personal Ubiquitous Comput.*, 5(1):50–53, 2001.

[18] T. Sohn and A. Dey. iCAP: an informal tool for interactive prototyping of context-aware applications. In *Proc. CHI (Conf. on Human Factors in Computing Systems)*, pages 974–975, New York, NY, USA, 2003. ACM.

[19] J. W. Spence. There has to be a better way! In *Proc. Agile Development Conference*, pages 272–278, Washington, DC, USA, 2005. IEEE Computer Society.

[20] D. ČubraniĆ, G. C. Murphy, J. Singer, and K. S. Booth. Learning from project history: a case study for software development. In *Proc. Conf. on Computer Supported Cooperative Work*, pages 82–91, New York, NY, USA, 2004. ACM.