Supporting the CANCEL Command Through Software Architecture

Len Bass
Bonnie E. John

December 2002

Architecture Tradeoff Analysis Initiative

Technical Note
CMU/SEI-2002-TN-021

Contents

List of Figures

Abstract

A system that supports the user's ability to cancel a command should be designed to achieve particular results. These results include the responses the system should make to the user, such as providing feedback to the user about the command's receipt, predicting the time the cancellation should take (for long-running cancellations), and indicating the state to which the system was returned after the completion of the cancellation. To support a cancellation command, a system should be designed so that the command is handled on a thread separate from that of the command being cancelled, the resources being used by the command being cancelled should be freed, and any processes collaborating with the command being cancelled should be informed of the cancellation. This note details the responsibilities that a system must implement to support command cancellation.

1 Introduction

For several years, the Software Engineering Institute (SEI) has been investigating how to couple architectural patterns with analysis for particular quality attributes. This investigation has been documented in several reports, for example, *Attribute–Based Architectural Styles* [Klein 99] and *Quality Attribute Design Primitives* [Bass 00]. The SEI has also been investigating the relationship between usability and software architecture [Bass 01]. The work summarized in this report is a combination of these two investigations. We describe an architectural pattern that supports the CANCEL command and discuss how to analyze it from a usability point of view. The scenario and its associated pattern were documented in *Achieving Usability Through Software Architecture* [Bass 01], but the analysis presented here has not been published previously.

2 The CANCEL Command

2.1 Problem Description

Consider the following scenario: a user initiates an operation on a computer system that takes more than one second to complete and the user changes his or her mind, wanting to return the system to its pre-operation state. Why the user initiated and then wanted to cancel the operation does not matter. It will happen in every system and every group of users. What does matter is that the system is equipped and designed to properly cancel operations that take more than one second.[1]

2.2 Stimulus/Response Measures

We characterize the important stimuli and their measurable, controllable responses as follows:

- **Stimuli**: The user changes his or her mind about launching an operation and takes some action to cancel it, for example, pressing the cancel key.
- **Response:** The system acknowledges receipt of the CANCEL command in sufficient time to prevent the user from thinking that he or she did not press the key hard enough (e.g., within 150 milliseconds [ms][2]). The system stops the operation and returns to the state it was in before the user launched the original operation, in less or the same time it spent on the operation prior to the user pressing the cancel key.[3] If the CANCEL

[1] Nielsen presents rules of thumb for response times. One second is about the length of time where a user will notice a delay and it is difficult to maintain an uninterrupted flow of thought [Nielsen 93, pp. 135-137]. Thus, it could also be thought of as the lower limit on when a user has the opportunity to re-think an action and decide to cancel it.

[2] From the Model Human Processor (MHP) [Card 83], it would take Middleman at least 270 ms to press the key again (100 ms for the user to perceive no signal, plus 50 ms to recognize that this means the button click was not recorded, plus 50 ms to initiate a new button click, plus 70 ms to press the button again). However, to interrupt that process, the signal that the button had been clicked must have been present in the world before the initiation began; hence it must be responded to within 150 ms. This number is in the ballpark of the 100 ms offered by Nielsen [Nielsen 93, p. 135], which allows the user to feel that the system is responding "instantaneously."

[3] This duration was determined by reasoning about the task, not empirical evidence about users' expectations in the real world. Our thinking is that it is reasonable to expect that operations will take approximately the same time to undo as they take to do initially, like walking from one place to another. Of course there are many operations that do not adhere to this rule (like sewing a seam and painstakingly picking out the stitches so as not to ruin the fabric), but it seems to be a reasonable target. Specifications for the particular system being analyzed or designed should be set through task analysis or empirical data collection with users if this default is not used.

operation will take longer to complete than one second, progress feedback should be displayed to the user.[4] Therefore the measures are

- the amount of time it takes to acknowledge the CANCEL command (e.g., change the cursor or bring up a dialog box with a progress bar)
- the degree to which the system's state after the CANCEL operation matches the system's state before the original action was launched
- the amount of time it takes to return the system to its original state (i.e., perform the cancellation)
- the accuracy of the feedback to the user about the degree to which the system's state after the cancel operation matches the system's state before the original action was launched
- the accuracy of the feedback about the CANCEL operation's progress (if more than one second[4] is required to restore the system to its previous state)
- the salience of the feedback about the CANCEL operation's progress (if more than one second[4] is required to restore the system to its previous state)

2.3 Cancellation Architectural Pattern

In this section, we describe the pattern by first presenting the cancellation pattern's module decomposition and process views. We then present the sequence of activities that will be undertaken by the elements in the pattern and the knowledge that those elements must have. We use the terminology from the book *Documenting Software Architecture: Views and Beyond* [Clements 02]. That is, modules are code-based entities, components are runtime entities, and elements may be either.

2.3.1 Module Decomposition View

The module decomposition view of the cancellation pattern is shown in Figure 1. This pattern has the following modules together with their responsibilities:

- *active modules*. These modules are performing the activities that are to be cancelled. They must cooperate with the cancellation module to provide resource and collaboration information. They also must have a mechanism for retaining sufficient information about the system's state prior to the invocation of the active module to be able to restore that state at any time. The mechanism will be exercised by the cancellation module, but the active modules must prepare resources so that the cancellation module can, in fact, exercise the mechanisms.
- *listen-for-cancellation module*. This module listens for the user to request the cancellation of the active modules. It must inform the user of the reception of the cancellation request and then informs the cancellation module. If necessary, it may spawn a new thread to control the operation of the cancellation module.

[4] Nielsen recommends that progress feedback come into play at about a second of delay time [Nielsen 93, p. 135]. He discusses using a "busy" cursor as feedback for delays less than 10 seconds and progress bars for delays greater than 10 seconds.

- *cancellation module.* This module must terminate the active thread, return the persistent resources to their state prior to the invocation of the active modules, release resources, provide feedback to the user about the progress and result of the cancellation, and inform collaborating modules of the termination of the active thread. It is responsible for gathering information about resources used by the active modules and about collaborating modules.
- *collaborating modules.* These modules are responsible for being receptive to information about the termination of the active modules.

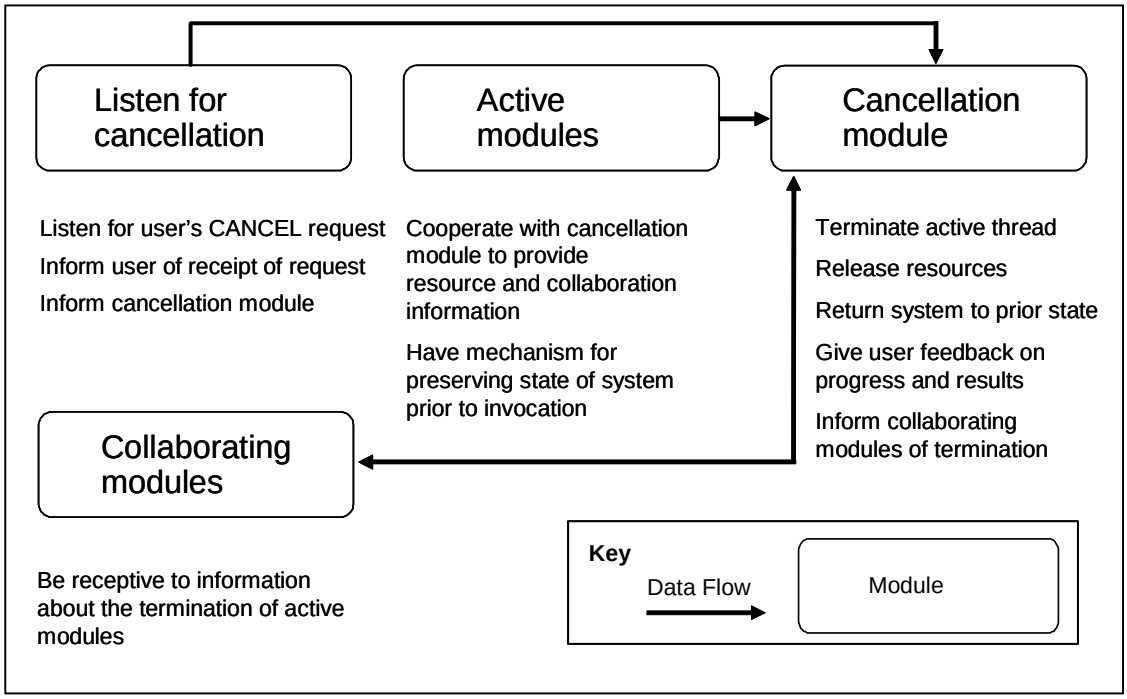

Figure 1: Module Decomposition View

2.3.2 Process View

Figure 2 presents the process view of the cancellation pattern. We assume the modules in the module decomposition view map directly onto components in the process view and maintain the same names. The following threads exist in the cancellation pattern:

- *the active thread.* This is a running thread that the user wishes to cancel. The activities running under its control of are the ones to be cancelled.
- *the listener thread.* This thread provides the user with a means to indicate what is to be cancelled.
- *the cancellation-control thread.* This thread manages the cancellation activities. It is independent from the active thread.
- *the collaborating-processes thread.* This may be a collection of threads, but we model it as a single thread. It manages those processes that collaborate with the active thread.

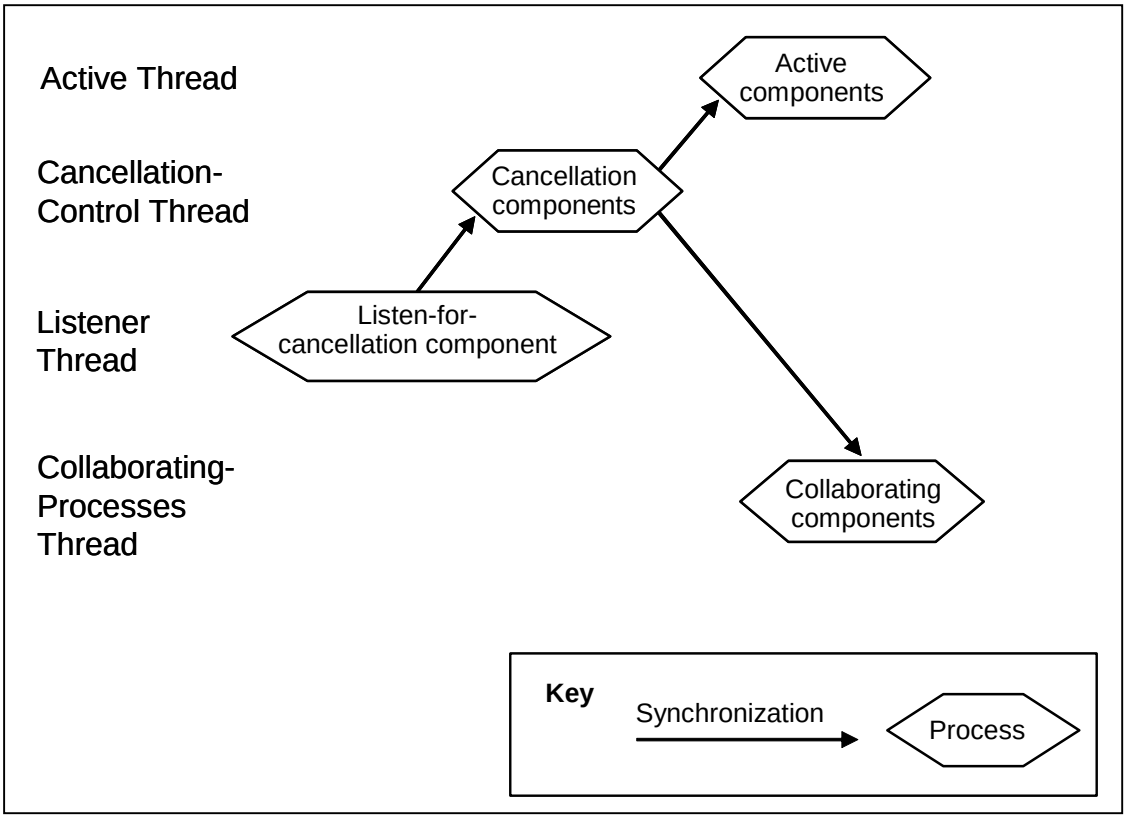

Figure 2: Process View of Cancellation Pattern

We explicitly label the processes in Figure 2 with the modules in the module decomposition view that map to them.

We now describe how these two views interact. The user sends a cancel stimulus to the listen-for-cancellation component running in the listener thread. This thread then

- provides feedback to the user that the cancel request has been received
- synchronizes with the cancellation component of the thread to be cancelled. This indicates to the cancellation component that the cancel activity should be carried out.

The cancellation component is executing under control of a thread created to carry out the cancellation. The active thread should not be expected to listen for a CANCEL request since it may be blocked for some reason, or it may be in an infinite loop.

The cancellation component carries out the following basic activities:

1. Terminate the active thread.
2. Inform the user of the progress and results of the cancellation.

3. Return the system to its state prior to the invocation of the active thread. This involves
   - restoring any persistent resources to their state before the invocation of the active thread
   - releasing resources acquired by the active thread
4. Inform threads collaborating with the active thread of the active threads' termination.

The first of these activities is straightforward. Presumably, the thread-control mechanisms of the operating system permit thread termination.

The second activity is also straightforward. The user should be informed about both the progress and the results of the cancellation.

The third activity (returning to the prior state) requires that the cancellation component be aware of the mechanisms for restoring persistent resources to the state before the invocation of the active thread. The cancellation component must also be aware of the resources acquired by the active components. This awareness can be achieved by a variety of mechanisms. The active components can report the acquired resources to the cancellation component, the cancellation component can intercept requests for resources, or the resource managers can provide this information to the cancellation component.

The fourth activity (informing collaborating threads) also requires knowledge on the part of the cancellation component. The cancellation component must be informed of collaboration, either synchronization or data communication, with other threads. It doesn't necessarily need to be informed of the state of the collaboration. The cancellation component can inform each of the collaborating threads of the active thread's termination. Then it becomes the responsibility of the collaborating threads to perform the correct actions, including providing feedback to the user about the progress and results of these cancellation requests, if necessary for completion of the thread's cancellation requests. Providing this information may be complicated, depending on the type of collaboration and the extent to which the collaborating components depend on the completion of the active components. One possible action is to treat the information that the active component is being cancelled as a cancellation request for the collaborating components. In this case, a recursive use of the pattern will achieve the desired results. Other types of collaborations may not require cancellation. In any case, a decision must be made regarding the desired result after the collaboration components have been informed of the active components' cancellation.

3 Analysis

In Section 2.2, we enumerated six measures. The analysis below describes how they can be determined.

3.1 Acknowledgment Time

The system should acknowledge to the user that it has received the CANCEL command. This should be done quickly enough to prevent the user from re-issuing the command (i.e., within 150 ms).[5] The acknowledgement should be done by the listener module.

3.2 Return to Original System State

Returning to the original system state depends on three things: (1) the mechanism used by the active modules to provide for returning to the original state, (2) the determination of the resources being used by the active modules, and (3) the notification of collaborators.

To analyze these measures, the correctness of the return to the original system state must be determined using architecture walkthroughs. Their purpose is to determine

- the resources that are being used by the active modules. For each such resource, the mechanism that informs the cancellation module of the use (or freeing) of this resource should be verified.
- whether the cancellation module actually frees the resources used by the active component
- whether the active module has a mechanism to return the system to a desired state and whether the cancellation module has the knowledge to implement this mechanism. The desired state can be either the state prior to initiation of the active module or a subsequent state when the user has indicated a desire to preserve intermediate results (such as a user-initiated “save” during an edit session).

[5] From the MHP [Card 83], it would take Middleman at least 270 ms to press the key again (100 ms for the user to perceive no signal, plus 50 ms to recognize that this means the button click was not recorded, plus 50 ms to initiate a new button click, plus 70 ms to hit the button again). However, to interrupt that process, the signal that the button had been clicked must have been present in the world before the initiation began; hence it must be responded to within 150 ms. This number is in the ballpark of the 100 ms offered by Nielsen [Nielsen 93, p. 135], which allows the user to feel that the system is responding “instantaneously.”

- whether collaborating module are known to the cancellation component and whether they are, in fact, informed of the cancellation
- whether the collaborating module take the correct action when the active module are cancelled. This determination is potentially far reaching, and there must be a clear understanding of the desirable and achievable actions of the collaborating module before the correctness of those actions can be determined.

3.3 The Amount of Time Taken to Return to the Original State

The performance target for the time to return to the original state should be less than or equal to the amount of time that has passed since the user took the action that was cancelled.[6] The assignment of threads to processors will have an impact on this measure. Therefore, this needs to be considered in light of a performance model.

3.4 Accuracy of Feedback

The feedback to the user that the CANCEL operation is indeed being carried out and the results of the cancellation should be, at a minimum, meaningful and accurate. The aspect of meaningfulness is independent of architectural decisions, so it will not be discussed here. (See *Usability Engineering* by J. Nielsen [Nielsen 93, pp. 123-129] for a discussion of how to "speak the users' language.")

The accuracy of feedback about the degree to which the system has returned to the original state is critical in areas where inaccurate feedback could cause user error. However, it is not critical in areas where the user will not be able to influence anything. This measure should be analyzed using system-specific scenarios. For instance, a scenario for a document-processing system might be that a user mistakenly initiates a global replace on his boss's name, changing it from Mr. Champ to Mr. Chump, detects his error, and cancels the operation. If the system were to restore the body of the prose to Mr. Champ but not the headers or footers, the user could be severely embarrassed, or worse, if he were not informed of the incomplete cancellation and sent the file to a client. Another scenario might not be as severe or immediate, as follows. If the same cancellation failed to release some memory resources, the user could still work and save files, but such a leakage might eventually lead to a degradation of performance. These scenarios can become more complex with the existence of collaborating processes.

[6] This duration was determined by reasoning about the task, not empirical evidence about users' expectations in the real world. Our thinking is that it is reasonable to expect that operations will take approximately the same time to undo as they take to do in the first place, like walking from one place to another. Of course there are many operations that do not adhere to this rule (like sewing a seam and painstakingly picking out the stitches so as not to ruin the fabric), but it seems to be a reasonable target. If this default is not used, specifications for the particular system being analyzed or designed should be set through task analysis or empirical data collection with users.

With regard to the accuracy of progress feedback in progress bars or dialog boxes, this information is often used as an estimate of the time until the cancellation will be completed, regardless of whether the information is presented as an actual time estimate (e.g., "40 seconds remaining") or as a percent-done indicator (where the user tends to expect a linear process). In either case, the system does not need to be more accurate at predicting the time to completion than people are at perceiving time duration. An exact quantitative specification cannot be pinpointed without knowledge of the specific task and user population. However, without context-dependent information, and erring on the side of over-predicting rather than under-predicting, we believe that most users would find time estimates within 20% of the actual elapsed time acceptable.[7] If these time estimates are highly variable, consider using a different form of feedback rather than progress bars or percent-done indicators, which both imply certainty.[8]

3.5 Salience of Feedback

The feedback should also be appropriately noticeable, or salient, where appropriateness depends on the amount of delay time and often on the task that the user is doing. Saliency has possible implications for this architectural pattern, so we give examples of analyzing it in this section. Some dimensions that make feedback more salient are the choice between visual and auditory feedback, the size of visual feedback, the volume of auditory feedback, and the preemptive nature of the feedback. The extensive research about alarms in the human-performance literature is relevant to the design of this feedback [Boff 88, Section 11.401].

At a minimum, the type of feedback depends on the amount of time the CANCEL operation will take. According to Nielsen, a percent-done indicator is usually appropriate to indicate delays greater than 10 seconds, but a "busy" cursor (e.g., a watch or hourglass, which is common on personal computers [PCs]) is more appropriate for delays under 10 seconds because it does not present the users with more information than they can comprehend in the time it is displayed [Nielsen 93, pp. 136-137]. (That is, for delays of less than 10 seconds, the CANCEL operation may finish and remove a percent-done indicator from the display before users can understand what the indicator was indicating, making them wonder if they missed some important information.) These recommendations were for generic office-like systems

[7] Boff reports results where people overestimate the duration of a several-minute-long time interval by between 10% and 60% [Boff 88, Sections 2.403 and 2.504]. Such results are influenced by the length of the interval, the experience of the user, and the complexity of what's happening during that interval, both externally in the world and internally to the person [Zakay 97]. Boltz reports that people are more likely to perceive an interval to be longer if they expected a shorter duration than if they expected a longer one; a practical application of this may be to err on the side of over-predicting the time estimate fed back to the user rather than under-predicting it, because the user might then perceive the delay as shorter in duration [Boltz 93].

[8] Alternatives might include a message saying that the time duration cannot be estimated accurately (stating why in terms the user can understand) and that the user can request notification in several ways (e.g., an auditory signal or a dialog box that appears on top of whatever else the user is doing or that stays on the screen until the operation is complete).

doing operations other than cancellation. However, the CANCEL command has special properties that deserve consideration, and systems other than office-work graphical user interfaces (GUIs) may have special properties as well, which should be explored through scenarios. For instance, consider the following scenarios.

Salience Scenario 1: Long-Duration Cancellation of a Low-Priority Task

The user initiates an operation that will take a relatively long time to complete, say 30 minutes. Assume that about halfway through, the user realizes that he or she set a parameter incorrectly and cancels the operation. The feedback says that the cancellation is estimated to take about 20 minutes to restore the original state. The user has other work to do that is independent of the operation being cancelled, so he or she wants to work in a different application on the same computer. The following questions should be asked about the salience of the feedback:

- *Does your architecture ensure that the progress feedback will be seen?*

  The feedback must be displayed prominently and long enough for users to read and comprehend it so that they can decide whether to start working on something else. Typically, this means that a preemptive dialog box showing the estimated duration and floating above every other window appears immediately after the cancellation is issued.

- *Does your architecture allow the feedback and the cancellation itself to be dismissed from the foreground so that the user can work on something else?*
- *How is the completion of the cancellation process signaled to the user?*

  Since the scenario assumes that this is a low-priority task, the feedback does not need to be preemptive, but instead could appear in a toolbar (e.g., some mail programs wave a small flag on a mailbox when a new message comes in). If so, movement or flashing will attract the users' attention in peripheral vision [Boff 88, Section 11.401-11.421], and an auditory signal could also be used. These attention-getting options should be easy to change if users prefer not to be interrupted but rather want to look deliberately for the feedback when they are finished with the other task.

Salience Scenario 2: Safety-Critical Cancellation

The user initiates an operation and then cancels it. The safety of the system critically depends on the system being in a stable state; therefore the user should not be able to initiate any other operations until the cancellation has been completed, no matter how long it takes. The following questions should be asked about the salience of the feedback:

- *Does your architecture ensure that the progress feedback will be seen?*

  The feedback must be displayed prominently and long enough for users to read and comprehend it so that they know not to try to do anything else until the cancellation is complete. Typically, this means that a preemptive dialog box showing the estimated duration and floating above every other window appears immediately after the cancellation is issued.

- *Does your architecture prohibit the user from doing anything else until the cancellation is complete?*

  Unlike the previous scenario, this dialog box should not be dismissible, ensuring that the cancellation is complete before the user proceeds.

- *How is the completion of the cancellation process signaled to the user?*

  Since the scenario assumes that this is a high-priority task that preempts other computer use, the completion feedback should probably interrupt the user fairly emphatically. Since the user may not have been paying attention to the computer (because the computer couldn't be used until the cancellation was complete), consider using an auditory signal to alert the user. Certainly the preemptive dialog box should display the results of the cancellation and the assurance that other operations are now safe to initiate.

References

[Bass 00] Bass, L.; Klein, M.; & Bachmann, F. *Quality Attribute Design Primitives* (CMU/SEI-2000-TN-017, ADA392284). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tn017.html>.

[Bass 01] Bass, L.; John, B.; & Kates, J. *Achieving Usability Through Software Architecture* (CMU/SEI-2001-TR-005, ADA393059). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tr005.html>.

[Boff 88] Boff, K. R. & Lincoln, J. E. (editors). *Engineering Data Compendium: Human Perception and Performance.* Wright-Patterson Air Force Base, Ohio: Harry G. Armstrong Aerospace Medical Research Laboratory, 1988.

[Boltz 93] Boltz, M. G. "Time Estimation and Expectancies." *Memory and Cognition 21,* 6 (Nov. 1993): 853-863.

[Card 83] Card, S. K.; Moran, T. P.; & Newell, A. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

[Clements 02] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architecture: Views and Beyond*. Boston, MA: Addison Wesley, 2002.

[Klein 99] Klein, M. & Kazman, R. *Attribute-Based Architectural Styles* (CMU/SEI-1999-TR-022, ADA371802). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. <http://www.sei.cmu.edu/publications/documents/99.reports/99tr022/99tr022abstract.html>.

[Nielsen 93] Nielsen, J. *Usability Engineering*. San Francisco, CA: Morgan Kaufman Publishers Inc, 1993.

[Zakay 97] Zakay, D. & Block, R. A. "Temporal Cognition." *Current Directions in Psychological Science 6,* 1 (Feb. 1997): 12-16.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2002	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Supporting the CANCEL Command Through Software Architecture	F19628-00-C-0003

6. AUTHOR(S)

Len Bass, Bonnie E. John

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2002-TN-021

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

A system that supports the user's ability to cancel a command should be designed to achieve particular results. These results include the responses the system should make to the user, such as providing feedback to the user about the command's receipt, predicting the time the cancellation should take (for long-running cancellations), and indicating the state to which the system was returned after the completion of the cancellation. To support a cancellation command, a system should be designed so that the command is handled on a thread separate from that of the command being cancelled, the resources being used by the command being cancelled should be freed, and any processes collaborating with the command being cancelled should be informed of the cancellation. This note details the responsibilities that a system must implement to support command cancellation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
architectural patterns, cancel command, quality attributes, software architecture	20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL