In *Proceedings of the 10th International Conference on Software Engineering & Knowledge Engineering (SEKE '98)*

Information Management for Release-based Software Evolution Using EMMA

Daryl McCullough (daryl@cogentex.com), Tanya Korelsky, Michael White

CoGenTex, Inc. (http://www.cogentex.com)

Abstract

This paper describes EMMA (Evolution-Memory Management Assistant), a tool to provide collaborative support for the release-based Evolutionary Delivery Model [4] of system development. Evolution information in EMMA is organized around four interrelated concepts: (1) properties, (2) assumptions, (3) solutions, and (4) change events.

This information provides context to relate development decisions to higher-level purposes and to the assumptions under which decisions are made. By managing information about design decisions and directions for change, EMMA helps project managers to understand the issues involved and to be proactive in planning for system evolution.

1 The EMMA Metamodel for System Evolution

EMMA, an Evolution-Memory Management Assistant, is a collaborative information management tool being developed by CoGenTex, Inc. under the sponsorship of the Evolutionary Development of Complex Software (EDCS) program[1] to address the needs of managers and developers when developing and maintaining complex, evolvable systems. (As described in [8], the costs of maintenance can be much, much more than the cost of initial system development. However, relatively little tool support exists for this crucial activity.)

EMMA supports the process of exploring, choosing among, and implementing alternative solutions by aiding in the acquisition and management of the information that needs to be shared among the project participants.

1.1 Goals and Solutions

The organizing principle behind EMMA's metamodel for system development is that to reason about system evolution, it is necessary to understand in detail the ***goal*** for a system and the ***solution*** constructed to address those goals. A goal for us is the list of desired properties that the system should have, together with the ***context*** (assumptions, definitions, resources and constraints) under which the system is to be developed.

A solution in the EMMA metamodel is an approach to satisfying some goal by either explicitly providing a reference to an actual implementation, or by partitioning system responsibilities among system elements, each of which is a solution for a simpler goal. For example, a solution for building a complex system might involve partitioning the system into user interface, data structures, and algorithms, each of which will be a solution with its own requirements to fulfill. In this way, complex high-level solutions are decomposed into simpler, more specific low-level solutions. Then higher-level solutions are constructed by composing low-level solutions.

The collection of goals, solutions, subsolutions and their associated system elements will be referred to as a ***solution structure.***

1.1.1 Solution Details.

Each solution node has the following information:

Type: The type of the solution, for example, "application", "documentation", "design", etc.
Description: An informal free-text description of the solution.
Alternative Solutions: As described above, each goal has a number of alternative solutions associated with it. One solution will be designated the ***active*** solution.
Justification: This describes the rationale for choosing the active solution from the set of alternative solutions, and explains how the subsolutions will be combined to form a solution to the goal.
Releases: A list of named releases for which the solution is relevant.
Status: Information about the state of development of this solution. Is it completed? Are all of its functional properties implemented? If not, which ones are completed

1.2 Properties and Context: Different Kinds of Requirements

In evolving the system, managers and developers need to keep in mind two different kinds of information:

1. **properties** of the desired system — what functionality the users of the system (where the

[1] Rome Laboratory Contract F30602-96-C-0220

"users" might not be human; they might be other systems) can expect the system to provide, and
2. **context** — what assumptions about users, other systems, computational infrastructure and operating environment the system can rely on.

The importance of distinguishing between assumptions and properties in system requirements has been taken into account in past work, such as the Requirements Specification Language developed by the Carnegie Group for the Armstrong Laboratory, Logistics Research Division, Wright-Patterson AFB, Ohio [6].

Properties and context elements of one node of the solution structure can be linked to properties and context elements of other nodes. In this way, dependencies can be traced in order to assess the impact of changes to requirements or assumptions.

1.2.1 What If?

In developing a schedule of releases, and deciding what functionality should be promised for what releases, a manager may explore the consequences of including or excluding a certain feature in a release, to see what impact such a decision may have on system development. The dependencies between properties and context information throughout an EMMA solution structure allow a manager to ask such "What if?" questions. trying to develop a schedule of releases. This capability will be described in Section 2.3.1.

1.3 Collaboration: Solution Owners and Solution Collaborators

Another important aspect of software development supported by EMMA's metamodel is collaboration. In developing a high-level solution, managers and developers proceed by factoring each solution into smaller subsolutions. Often, developers working on different parts of the solution must coordinate their actions. To facilitate this coordination, the EMMA metamodel has a notion of the ***owners*** and ***collaborators*** for solutions. The owner of a solution is responsible for that solution and any solution developed to implement it. A collaborator for a solution is a developer who is working towards implementing the solution. In the EMMA metamodel, to support accountability, we stipulate that only a solution owner may change the attributes associated with the solution, but that a collaborator for a solution is able to create, modify, and act on a plan for implementing the solution.

1.4 EMMA and Evolutionary Development Process

How can the solution structure with its network of dependencies described above be used in software development and evolution?

While modern software development methodologies do a fairly good job of making explicit the requirements that the system and its components must meet, there is comparatively little support for recording the thoughts of customers and developers about likely directions of evolution for the system.

In the evolution of a system from one release to the next, several changes may take place:

1. More planned functionality may be incorporated. The early releases of a system may only satisfy a subset of the system requirements, while later releases will gradually cover more and more of the requirements for the ultimate system. (The phenomenon of "scoping out" of requirements in early releases is described in [11].)
2. The requirements may change, due to user feedback from earlier releases.
3. The available external resources (tools, languages, and infrastructure) may change.
4. The approach to implementation (plans) may change to reflect changes of requirements or resources.

EMMA provides a mechanism for recording these kind anticipated changes. This kind of information can provide a valuable head start for future system developers charged with responding to such changes.

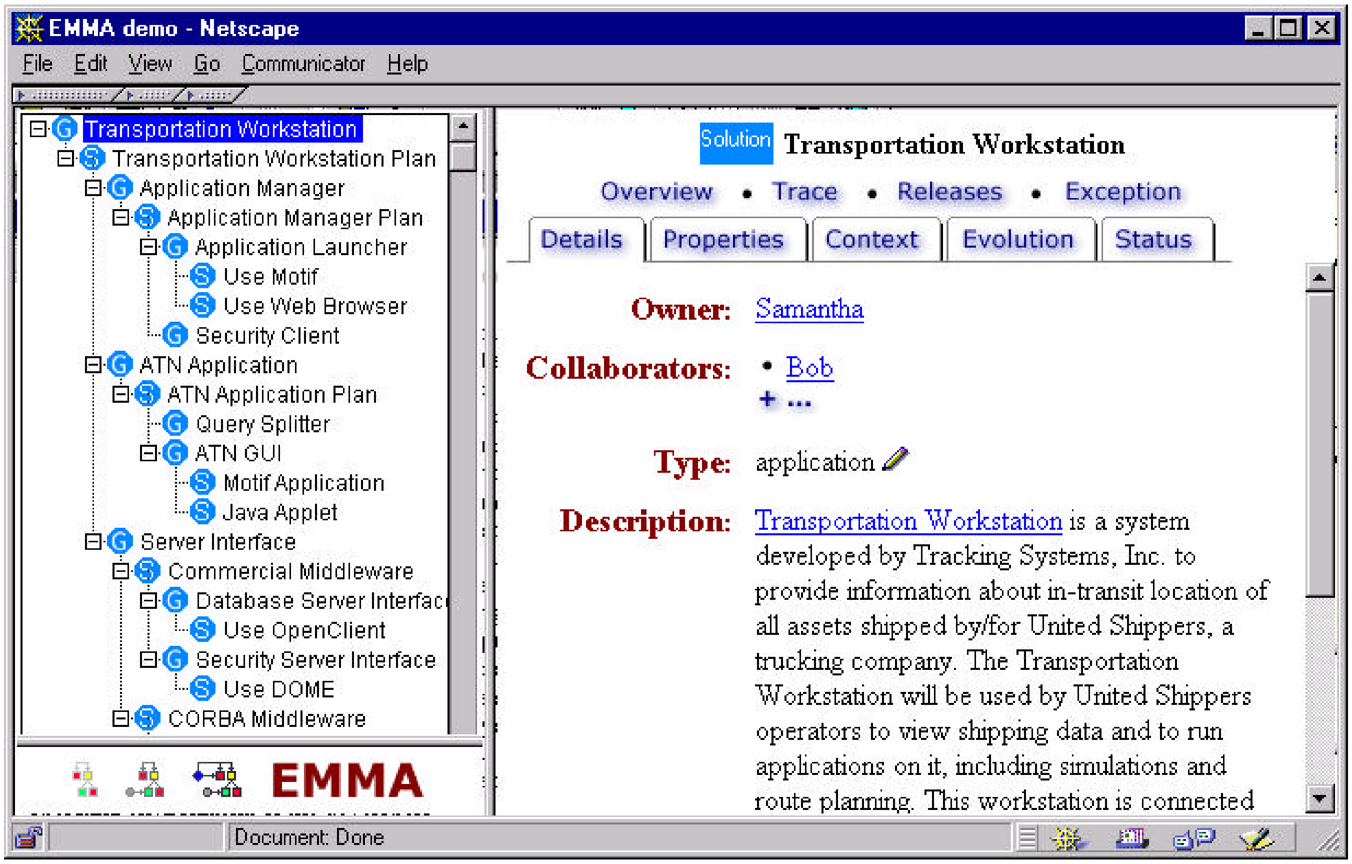
1.4.1 Dynamic Changes.

The EMMA solution structure for a system records, in a stable structure, both the current structure of a system solution and future plans for the evolution of this structure. In addition to this static structure, the EMMA metamodel also supports dynamic changes to plans and solutions. There are two types of change supported by EMMA: top-down change, due to a change of requirements, or a change in the release plan, and bottom-up changes (or "feedback" from developers), due to circumstances arising in implementing some low-level solution. For top-down changes, EMMA uses the dependencies between high- and low-level properties in order to trace the effects of changes in the assignment of properties of releases. This feature will be illustrated in the scenario in Section 2.3.

2 Example Scenario: Release Planning for Transportation Workstation

Our example to illustrate the use of EMMA is a hypothetical system to provide in-transit visibility to supplies shipped around the country. (The first release of the EMMA prototype will also have solution structure for EMMA itself and for other on-going projects at CoGenTex.) This system consist of an extensible collection of feeder systems providing data on the progress of shipments, a replicated database for this data, and a number of applications that provide views and metrics on this data. The Transportation Client Workstation is used to view transportation data and run applications on it. This workstation is connected to a security server to check the identities and authorizations of the operators.

Figure 1 below shows an EMMA screen for a solution structure corresponding to this workstation. On the left part of the screen, a tree shows the structure of solutions and plans for the Transportation Client Workstation. This tree allows the EMMA user to navigate through the solution structure. For any solution or plan selected on the left, the right part of the screen shows the associated information about that node. This information is partitioned into ***Details, Properties, Context, Evolution,*** and ***Status*** pages. The Details page for a solution gives information about collaborators and informal descriptions of the solution nodes. The Properties, Context and Evolution pages for a solution give the information about these aspects of a solution (described in Section 2). The Status page tells for which releases a solution is complete, and also lists any unhandled events.



Solution **Transportation Workstation**

Owner: Samantha

Collaborators:
- Bob
- + ...

Type: application

Description: Transportation Workstation is a system developed by Tracking Systems, Inc. to provide information about in-transit location of all assets shipped by/for United Shippers, a trucking company. The Transportation Workstation will be used by United Shippers operators to view shipping data and to run applications on it, including simulations and route planning. This workstation is connected

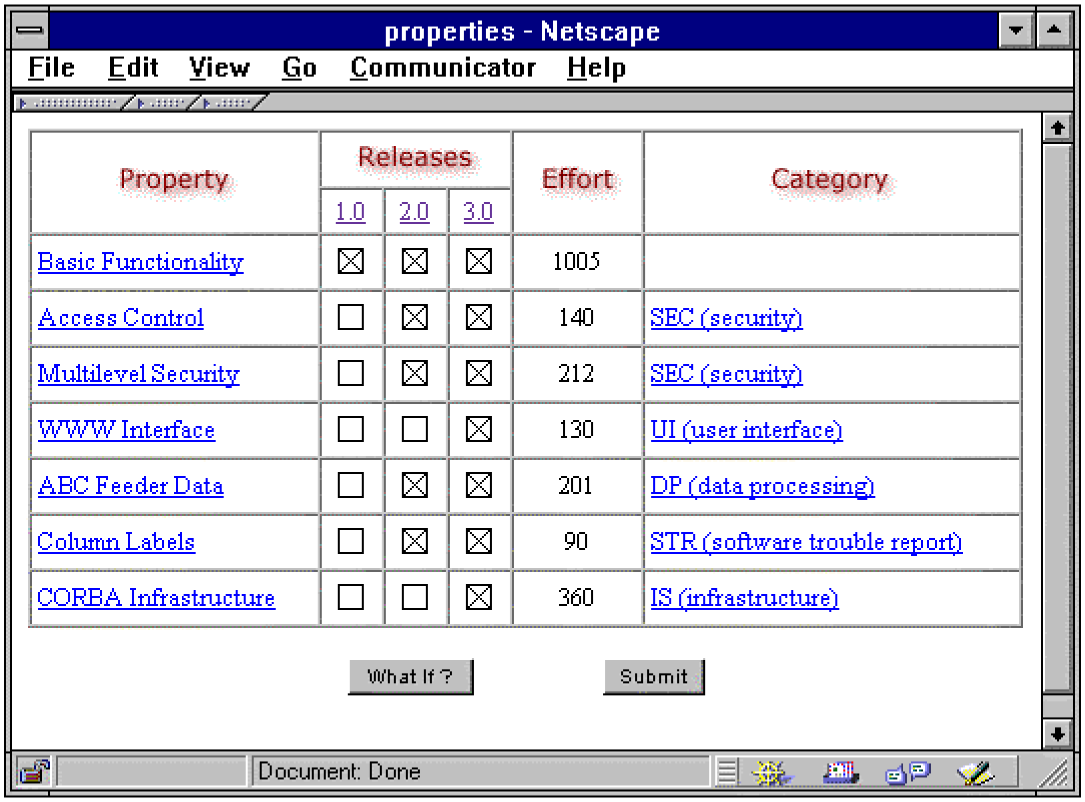
Figure 1. A Sample EMMA Page

As shown in the figure, the top-level goal is to build the Transportation Client Workstation. Under this solution are three subsolutions, each with its own goal, corresponding to components of the Transportation Client Workstation: (1) the application manager, which handles interaction with the user and launches applications; (2) the ITV application, which handles intransit visibility queries; (3) the server interface, which communicates with various data servers for transportation and security data.

At deeper levels, some goals have a number of alternative solutions. For example, the Server Interface has two alternative solutions: Commercial Middleware, which uses commercial interfaces that are not CORBA-compliant, and CORBA Middleware, which uses specially built CORBA-compliant interfaces. Alternatives in the solution structure represent possible future upgrade paths that involve architectural changes. In this example system, there are two planned major upgrades that involve changes of architecture: (1) making the server interfaces CORBA-compliant, and (2) making the application launcher and user-interface World-Wide-Web compliant (by switching to Java and HTML).

2.1 The Current Release Plan

The release plan for the system is captured on the Properties page of the top-level solution (Figure 2). This page shows that there are three planned releases for the Transportation Client Workstation. Basic functionality will be available in all three releases. For release 2.0, some additional features — Access Control, Multilevel Security and ABC Feeder Data — will be added. In addition, in release 2.0, a software trouble report (reflecting complaints by users) having to do with Column Labels will be addressed. In release 3.0, the system will be upgraded to have a WWW interface and to be CORBA compliant. We will illustrate later how this release can be changed by circumstances.



Property	Releases			Effort	Category
	1.0	2.0	3.0		
Basic Functionality	☒	☒	☒	1005	
Access Control	☐	☒	☒	140	SEC (security)
Multilevel Security	☐	☒	☒	212	SEC (security)
WWW Interface	☐	☐	☒	130	UI (user interface)
ABC Feeder Data	☐	☒	☒	201	DP (data processing)
Column Labels	☐	☒	☒	90	STR (software trouble report)
CORBA Infrastructure	☐	☐	☒	360	IS (infrastructure)

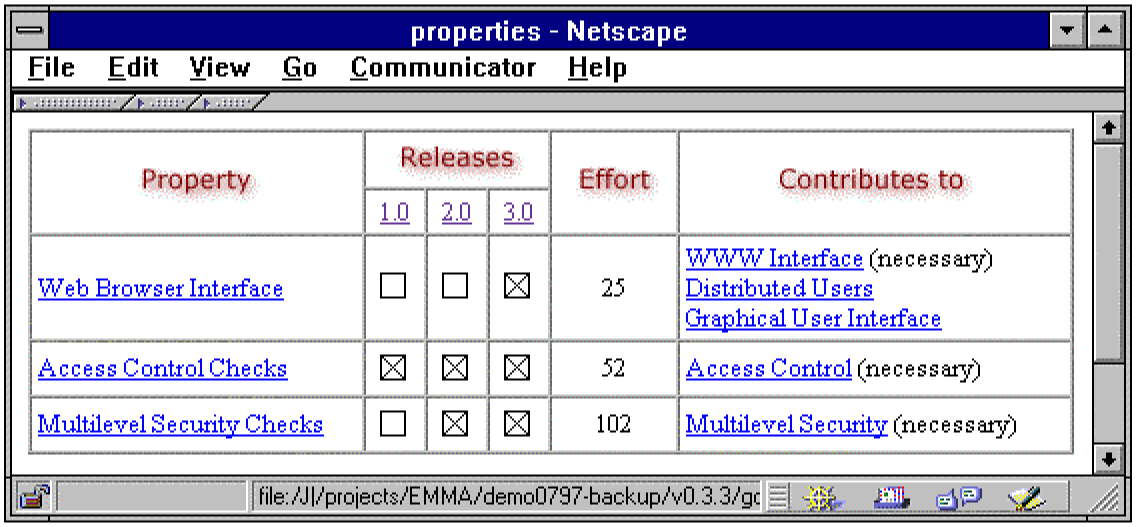
Figure 2. Properties for the Top Solution

2.2 Relating Low-level Properties to High-level Properties

The Properties page for the lower-level solution Application Launcher is shown in (Figure 3). The Application Launcher starts up a Transportation application, provided that the operator's identity has already been authenticated and the operator has permission to run that application.

In addition to the release information and the effort estimates that are present in the top-level Properties page, each property of a lower-level solution has a list of properties of higher-level solutions to which it contributes. If a low-level property is necessary to implement a higher-level property, then this is also indicated.

In EMMA, information about properties flows both upwards and downwards. The effort estimate for a high-level property is computed by summing all the effort estimates of the necessary low-level properties that contribute to it. (The user can override this computed estimate.) In the opposite direction, the release information for low-level properties is checked automatically from the release information for high-level properties: if a low-level property is necessary for a higher-level property, then this low-level property must be implemented at least by the release in which the high-level property is implemented.


properties - Netscape
File Edit View Go Communicator Help
file:/J|/projects/EMMA/demo0797-backup/v0.3.3/gc

Property	Releases			Effort	Contributes to
	1.0	2.0	3.0		
Web Browser Interface	☐	☐	☒	25	WWW Interface (necessary) Distributed Users Graphical User Interface
Access Control Checks	☒	☒	☒	52	Access Control (necessary)
Multilevel Security Checks	☐	☒	☒	102	Multilevel Security (necessary)

Figure 3. Properties Page for a Lower-level Solution

2.3 *Sending Feedback*

The EMMA solution structure represents the planned development path for a system. However, things do not always go as planned, so EMMA provides mechanisms for changing plans. The most innovative aspect of EMMA's support for change is through its feedback mechanism. In our example, we will suppose that Release1.0 has already been completed, and that the development team for the Transportation Client Workstation is working on Release 2.0. John, the owner of the solution Application Launcher, runs into a problem in implementing the properties that are planned for this release. We will suppose that the approach that John is taking to implement Access Control Checks requires some security functionality that is not available in the current infrastructure, but is available in a recent version of the Netscape web browser. This creates a dilemma for John, because the current plan is not to switch to using Netscape until Release 3.0 (when the top-level feature WWW-Interface comes into play).

To officially notify others of his problem, John selects the button labeled Feedback on the EMMA page for his solution. This causes a form to pop up requesting information about the problem and John's suggestions (if he has any) for fixing it. In filling out the feedback form, John may supply both an informal description of the problem and also a structured EMMA-understandable description of his proposed solution. In this case, he suggests that the top-level feature WWW Interface be moved up to Release 2.0 (it was originally scheduled for Release 3.0).

2.3.1 Handling the Feedback: Changing the Release Plan

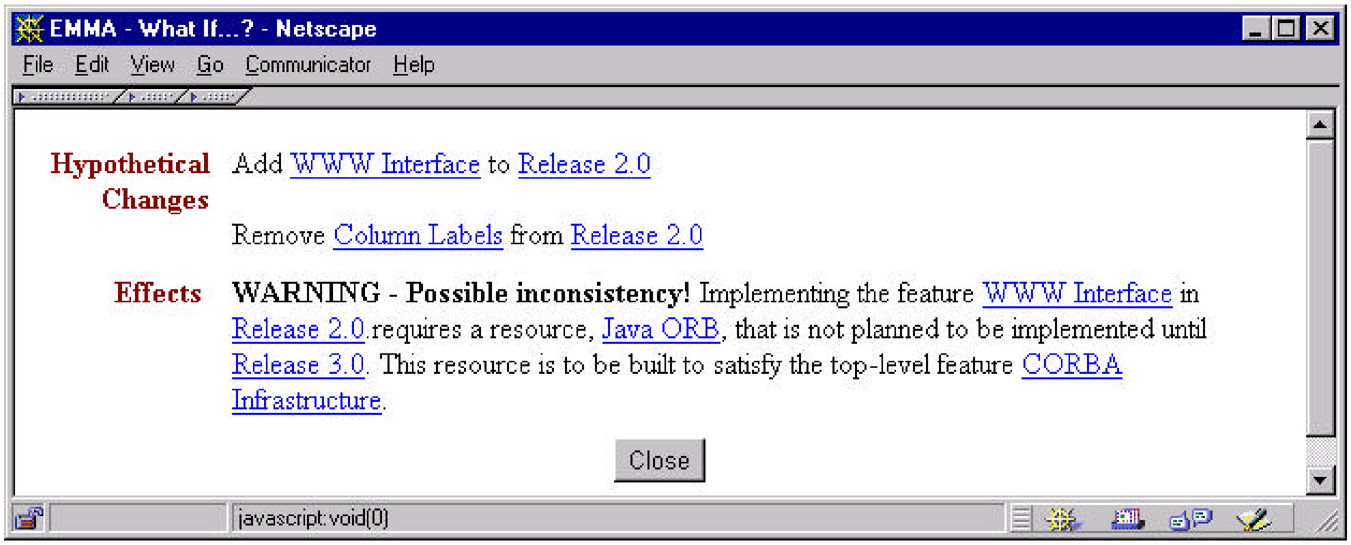
Similar to handling exceptions in programming languages, EMMA feedback posted at one level of a solution structure may either be handled at the next higher level, or it may be propagated upward. If it is not handled at any intermediate level, then the feedback may lead to a change of plans for the system as a whole. In our example, we will suppose that the feedback propagates all the way up to the top solution, where it becomes a problem for Samantha, the owner of that solution (who might be the manager for the project).

EMMA informs Samantha that she has unhandled feedback, which she views from her Status page. She reads the description of John's problem, and considers possible actions to address it. John's suggestion is that the implementation of the WWW-Interface be moved up into Release 2.0, so Samantha goes to her Properties page to see if that is a realistic option. As shown in Figure 2, the property WWW-Interface is currently scheduled for Release 3.0. Samantha changes this by checking the column for Release 2.0 on the row for WWW-Interface. Since this property requires an estimated effort of 130 person-hours, implementing this feature would delay Release 2.0 unless something else is dropped. Samantha therefore removes the feature Column Labels from Release 2.0 to compensate (this saves 90 person-hours, which roughly compensates for the additional work).

When Samantha is done with her changes, she clicks on the button labeled "What If?". This causes EMMA to compute the anticipated effects of this change and to report any possible problems that it discovers. The results for Samantha are displayed in Figure 4.

This summary — which in the EMMA prototype is automatically generated from EMMA solution structure data, using CoGenTex' natural language generation technology [7, 9]— shows that the proposed release plan contains a possible inconsistency. The problem with implementing a WWW Interface in Release 2.0 is that one of the solutions necessary for implementing the feature requires a resource (Java ORB) that will not be available until Release 3.0. The summary also provides the information that this resource is to implement the top-level property CORBA Infrastructure.

EMMA discovers the problem in the release plan by combining two kinds of traces: (1) downward tracing following the "contributes-to" relationship to discover what properties of what solutions are important for implementing the top-level property, and (2) sibling-level tracing to see what solutions produce resources that are required for which other solutions (this information is found in the Context for each solution).


EMMA - What If...? - Netscape
File Edit View Go Communicator Help
Close
javascript:void(0)

Hypothetical Changes Add WWW Interface to Release 2.0

Remove Column Labels from Release 2.0

Effects **WARNING - Possible inconsistency!** Implementing the feature WWW Interface in Release 2.0 requires a resource, Java ORB, that is not planned to be implemented until Release 3.0. This resource is to be built to satisfy the top-level feature CORBA Infrastructure.

Figure 4. "What If?" Results

2.4 Recording the Rationale for the Change

If Samantha (and, presumably, her customer) is satisfied with the new release plan, she can make it official by selecting the button labeled "Submit" on the Properties page. This causes EMMA to display a form asking for the rationale for the change. EMMA records this information so that in future development of the system, developers can better understand the reasons for past design decisions about the system solution structure. (The need for recording rationale for changes was noted in [3].)

3 Related Work

The EMMA metamodel has some similarities with other approaches to recording and reasoning about assumptions and solutions for systems and system components. EMMA's approach to system development by partitioning system solutions into solution components, each of which must satisfy certain requirements, is similar to the idea of "Design by Contract" [5].

Each solution in an EMMA solution structure represents functionality that the collaborator "promises" to provide (by building a solution). The context associated with a solution represents "commitments" made by the solution owner (which can reflect functionality provided by other collaborators, or "guarantees" about the users or operating environment). This view of system development is compatible with (but not limited to) the concept of ***required*** and ***provided*** interfaces for components [1, 2, 10].

4 Conclusions

EMMA is an innovative experimental prototype under development, which is designed to support system development and evolution in the following ways:

1. It supports structured, solution-directed communication and collaboration about system evolution. EMMA provides a framework for recording information about the expectations and responsibilities of all members of the development team.
2. EMMA presents information about the solution status of a development project, tailored for each developer. This solution status information includes uncompleted solutions and solutions that have become inconsistent because of changes to functional requirements or assumptions.
3. EMMA provides evolution assistance in several ways:
   - 3.1. It provides a mechanism for "anticipations". A developer can anticipate future changes in requirements, assumptions and resources, and can use Emma to record an alternative plan for responding to these changes. These recorded anticipations can provide a valuable head start for future system developers charged with responding to such changes.
   - 3.2. It provides a new feature for "posting feedback". This is a mechanism that allows for high-level requirements or assumptions to be changed to take into account problems and opportunities encountered in implementation. A developer who runs into problems in implementing a solution or encounters unforeseen opportunities can post feedback to request a modification to the solution plan that addresses the problem or takes advantage of the opportunity.

We believe that for systems that are meant to be long-lived and evolving, a tool like EMMA should be used from the beginning of the system development and throughout their life cycle. Without the convenient support for evolution-oriented collaboration and information management in the early stages of a system's development, it is very unlikely that the developers and managers will be motivated to document the sort of information that will be needed for future evolution and maintenance.

Acknowledgements

The authors would like to thank DARPA and Rome Laboratory, especially Carla Burns, our project engineer, for supporting this work, Lyn Uzzle of Lockheed-Martin for suggesting the specialization area of release planning and for general discussions, Frank Svoboda of Lockheed-Martin for his evolutionary example, David Post of CoGenTex for his suggestions and for his work on the implementation of the prototype, Ted Caldwell of CoGenTex for contributing his Javascript, Exemplars, and user-interface expertise, and Owen Rambow for general discussions.

5 Bibliography

1. CBSE. Ning, J. Q., Miriyala, K., and Kozaczynski, W., "An Architecture-driven, Business-specific, and Component-based Approach to Software Engineering," *Proceedings of the 3rd International Conference on Software Reusability,* Rio de Janeiro, Brazil, November 1994.

2. Euclid. B. W. Lampson, J. J. Horning, R. L. London, J. G. Mitchell, and G. L. Popek. "Report on the Programming Language Euclid," *ACM SIGPLAN Notices*, 12(2), February 1977.

3. Hughes. Falcioni, R., and Buvel, R., MODULAR EMBEDDED COMPUTER SOFTWARE (MECS), Tech. Report WL-TR-92-1113, produced by Hughes Aircraft Company for Wright Laboratory, Wright-Patterson AFB OH.

4. McConnell. Steve McConnell, *Rapid Development.* Microsoft Press, 1996.

5. Meyer. Meyer, B., "Applying 'Design by Contract'," *IEEE Computer*, October 1992.

6. Rapid-WS. A. Kott and J.L. Peasant, "Representation and Management of Requirements: the RAPID- WS Project", *Concurrent Engineering Research and Applications*, Volume III 1995.

7. RealPro.Benoit Lavoie and Owen Rambow, "RealPro ¾A Fast, Portable Sentence Realizer", *Proceedings of the Conference on Applied Natural Language Processing (ANLP'97),*1997, Washington, DC.

8. SoftwareMaintenance. Martin, J. and McClure, Carma. SOFTWARE MAINTENANCE: THE PROBLEM AND ITS SOLUTIONS, Prentice-Hall, Englewood Cliffs, NJ, 1983.

9. TextGeneration. Owen Rambow and Tanya Korelsky, "Applied Text Generation", *Third Conference on Applied Natural Language Processing,* pages 40-47, Trento, Italy, 1992.

10. TRP. Jin W. Chang and Colin T. Scott, "TRP Support Environment (TSE)," International Workshop on CSCW and the Web, 1996.

11. WinWin.Mingjune Lee. "Foundations of The Winwin Requirements Negotiation System". Technical Report TR-96-501, USC Center for Software Engineering, University of Southern California, University Park, Los Angeles, April 1996.