

Software Maintenance Assistant System

Koji Takeda, Mitsuyuki Inaba, David Chin, Kazuo Sugihara, Isao Miyamoto
Software Engineering Research Laboratory
Department of Information and Computer Sciences, University of Hawaii
Honolulu, HI 96822, U.S.A.

Abstract. A research aiming at development of practical solutions to software maintenance crisis was conducted by universities and computer companies. This paper reports Software Maintenance Assistant system (SMA), an outcome of the research effort, which provides (1) visualization language for complexity management, (2) methodology based user interface for structured maintenance process, and (3) automatic reverse and forward engineering tools to keep specifications and source code consistent.

1. Introduction

Maintenance programmers are overwhelmed by the complexity of source code. After a number of modifications, design documents are outdated or non-existent. There is no guide book. They just have to do it hoping the bugs disappear before the deadline. Despite of the fact that maintenance requires experience and skills, novice programmers are often assigned to this time-consuming task, making the situation worse.

Ever increasing maintenance cost made public aware of the importance of maintenance research. However, we have seen not too many researches aiming at creating practical solutions. Most of them were ideas built around reuse of source code based on object-oriented technology. What the maintenance practitioners need today are maintenance support tools which can reduce the complexity of source code, keep specifications and source code consistent, and enable more structured way of doing maintenance even by non-experienced programmers.

These requirements are satisfied in Software Maintenance Assistant system (SMA) developed by University of Hawaii, University of California at San Diego, and several computer companies for 1992 - 1994. In SMA, all maintenance activities take place on the visual models of the system, not on the source code. The visual models are automatically generated by model generation tools. After maintenance is completed, the source code is automatically re-generated. In this fashion, the source code and the visual models are always consistent. The visual models are represented by University of Hawaii's MERA language. MERA is a generalized Entity-Relationship model, and incorporates a number of features designed to control the complexity of models presented to the user. SMA's user interface allows or restricts access to maintenance tools and objects based on the information written in the maintenance methodology model, making it easier for non-experienced programmers.

Section 2 will explain various tools of SMA. MERA language will be explained in Section 3. The methodology model and SMA user interface will be explained in Section 4. All references to SMA tools are maintained online. The tool name written in square brackets in this text can be used to browse the references.

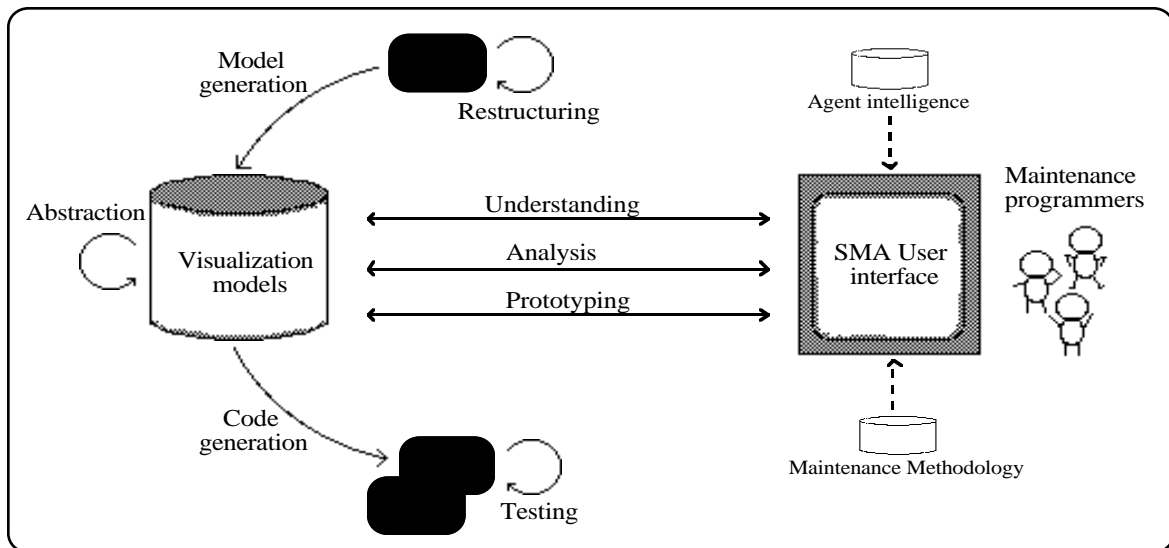


Figure 1. Overview.

2. SMA Tools

Restructuring. The restructuring tool automatically converts unstructured source code into structured code. A structured code produces better models [Restructuring Tool].

Model generation. All models generated by the model generators are stored in the object base. In addition to typical functions of version control and name management, the object base provides unique search functions based on a graph pattern matching algorithm [Object base]. The models generated are at the same level of abstraction as the source code. However, the source code is now described in multiple points of view by well-separated models. Total of 12 different models are generated each having specialized formalism for the best understandability [Model generator]. Also, automatic layout generator adds layouts to the models basing on the semantics of the model formalism [LYCA].

Abstraction. Program abstraction tools are then used to add abstractions to the models. One of the abstraction tools uses read/write statements analysis to create a dependency structure of the program [HPAS]. Another tool uses comment analysis techniques to perform abstractions [FAUCET]. Knowledge-based planning module simplifies usage of various tools for reverse engineering [PROUD]. After abstraction tools are used, programmers are more maneuverable with hierarchical representation of the system.

Understanding, Analysis, Prototyping. Visual models can be browsed and animated to aid understanding of the program. Then, various static and dynamic analysis tools are used to isolate the problem and develop solution [Petri Net analyzer] [GPAT]. Any changes to the program can be first analyzed for its impact to other parts [Impact analyzer]. If a new function is to be added, it can be done through a model editor. The editor takes information from the formalism file of each model and prohibits the user to create models which violates the formalism [MeraTalk]. A checking tool checks the models against a set of correctness criteria specified by first-order logic [Correctness checker]. If user interface is to be modified, user interface prototyper can be used to quickly design screen layout and dialog structure. This tool generates simulated user interface and allows the programmer to verify conformance to the requirements [UP].

Code generation. Modified models are then translated back to the source code. As mentioned earlier, the process is automatic, therefore consistency is guaranteed. By this time, the models contain enough abstractions, so they can serve as design specifications. SMA also provides a document conversion tool which generates textual version of the models

automatically [LDS].

Testing. A testing tool is provided for in-depth testing of the generated source code [Symbolic evaluator].

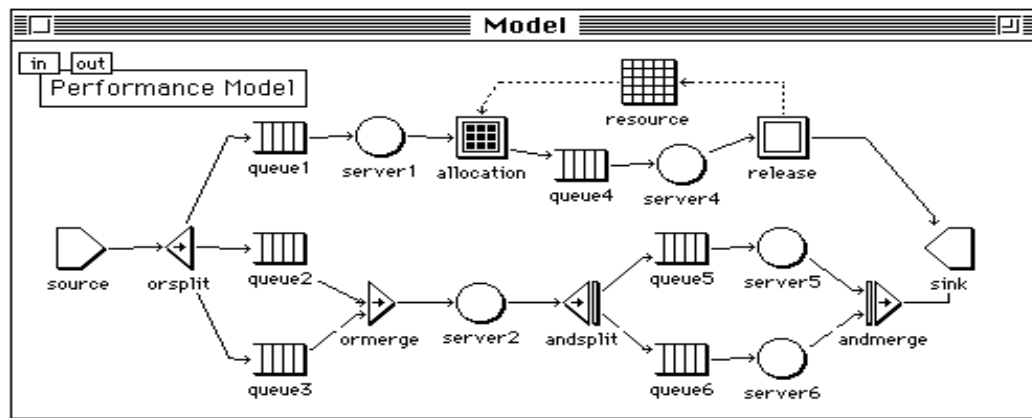


Figure 2. An example of MERA model.

3. MERA Language

MERA is a versatile graphical language developed at University of Hawaii. It is based on Entity-Relationship model, and it supports representation of various aspects of software. A unique feature of MERA is that it can be used to define many different graphical languages by creating a meta-model in MERA that describes the formalism of the specific graphical language. This approach is superior than having a superset of multiple languages which increases chance of the user being overwhelmed by the complexity of the language definition itself.

The goal of MERA in SMA is to provide ways to manage the complexity of a source code representation. This section describes MERA's complexity control features which are (1) view operation support, (2) abstraction support, (3) multiple view support, and (4) multimedia support.

First, MERA's view operation functions allow the user to manipulate the way of presenting models so that the user can focus on a certain aspect of the model. In MERA, definitional information such as attribute values and object names is clearly separated from view information such as location of objects and icon images. Separating definitional data from view data makes it easy to freely manipulate model appearance with the model definition intact. Examples of view operations include hiding irrelevant information and highlighting attributes having specific values.

Second, MERA provides two abstraction functions to organize type definitions in meta models and entities and relations in instance models. The first one, class abstraction, allows definition of a type as a subtype of another. This makes the language definition more structured, and assists user's comprehension. The second one, instance abstraction, allows definition of objects (entities or relations) as particular compositions consisting of a set of other objects. Thus models become more hierarchical and each layer becomes small enough to view without much scrolling.

Third, MERA supports modeling of software in multiple points of view. MERA is capable of specializing MERA into multiple languages having different formalisms. Multiple formalisms, each of which represents only a small coherent aspect of software, reduce complexity and improve understandability. A problem with multiple formalisms is that the formalisms often lack connections to one another, which leads to inter-model consistency problem. This problem is solved in MERA using linkage models which describe various types of correspondence between formalisms.

Finally, MERA's multimedia feature improves understandability of the model content. In addition to iconized entities, each object in a model can have an animation script. Such script

defines a visual effect. Examples of the visual effect are "highlight object", "vibrate object", etc. Visual effect helps to represent dynamic behavior of the target system which is often difficult to understand from a static representation. For example, an order of data flow is difficult to understand from a dataflow diagram. That can be effectively shown in animation of dot-flow in timely manner.

With these features described above, MERA provides the maintenance programmer with choices of working with the visualization at a desirable level of detailness and focusedness.

4. User Interfaces of SMA

SMA has three types of user interface. The first one is called process-oriented user interface which is suitable for a novice maintenance programmer. It can guide the programmer through maintenance activities by allowing access to only tools and products relevant to the current task. The second one is called product-oriented user interface which is suitable for a project manager. It provides quick access to the state of each product and overall project progress. The third one is called tool-oriented user interface which is suitable for an expert maintenance programmer. It provides direct access to tool functions and products anytime.

SMA user interface is model-driven, where each type of user interface is created by interpreting underlying maintenance methodology model in a specific view point. The methodology model consists of process model, product model and tool model. The process model describes a maintenance procedure. The product model describes various maintenance objects and project management objects. The tool model describes functions of available tools. These three models are linked by various inter-dependency relations such as "process P generates product X and product Y" and "tool T or tool Y can generate product Z". During the maintenance process, these models hold instance level information such as "process P1 generated product X1 at time t1 by user u1" and "tool Y was used to generate product Z1".

The methodology model is created by adapting a generic model to a specific maintenance project based on various environmental parameters. Such parameters include programming language and operating system of the target software, type of maintenance, available computer tools, project time line, and programmers. The methodology model is also written in MERA using methodology model formalism and MERA editor.

SMA also provides assistant capability. The maintenance programmer can ask questions about SMA system, search and view models, and even simulate changes through plain English dialog. This subsystem, called Maintenance Consultant (MC), is implemented as an agent. Through its user modeling capability, its own agenda and vast knowledge of English in maintenance domain, MC interprets intended meaning of the user's question and derives a plan for how to assist this user. MC communicates with another SMA agent which invokes and coordinates SMA tools, so that MC's user help will be more than just a presentation of information.

The methodology model and the model-driven user interface enable a structured approach to maintenance process. The agent-based assistant capability is great help to non-experienced programmers.

Descriptions of SMA tools appeared in this paper can be found in our home page:
<http://samurai.ics.hawaii.edu/>

Acknowledgement

We would like to thank our graduate assistants, visiting researchers, secretaries, and sponsors who all worked hard to make this dream project a reality: Mukherjee, Xiaomei, Ueda, Bin, Douglas, Kobayashi, Kato, Francois, Sahara, Philip, Michael, Raman, John, Di, Ya, Pareek, Govinda, Nadeem, Kuroko, Hiroko, Nishimura, Adachi, Shimada, Ishii, Nemoto, Natalie, Guangming, Yuhong, Sekhar, Bo, Yixin, Hai, Xiaobo, Khan, Xiandong, Yamamoto, Mayumi, Tsubaki, Yoga, Ree, Narsimha, Kamala, Bal, Ye, Haruka, Keiko, Yukari, Ethel, Fujitsu Ltd., PFU Ltd., and FHL Ltd.