IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP5225

This paper appears in *Managing Modern Organizations Through Information Technology*, Proceedings of the 2005 Information Resources Management Association International Conference, edited by Mehdi Khosrow-Pour.

Architectural Evaluation Support Tools

Phillip Schmidt, Greg Mulert and Zaven Petrosyan
The Aerospace Corporation M1-113 & M8-080, 2350 E. El Segundo Blvd., El Segundo, CA 90245,
{phillip.p.schmidt, gregory.l.mulert, zaven.petrosyan@aero.org}

ABSTRACT

The Software Architecture Representation Analysis Experimentation Environment (SARAEE) is an applied research project that investigates architectural representation issues that arose from our Real-time Embedded Architecture-Centric Testbed (REACT) experience. This report documents three architectural evaluation support tools that were developed by SARAEE to support on-going architectural analysis conducted by REACT. The tools include a requirement visualization tool that assists in the visualization of requirement dependencies, a tool to support REACT's aspect-oriented architectural assessment approach, and an early prototype tool that assists in investigating the potential impact of unanticipated concerns. This report presents the motivation for developing the tools to support REACT and their features. Finally, we discuss future directions in our tool development efforts.

INTRODUCTION

REACT is a facility used to conduct architectural assessments of real-time embedded systems to discover and resolve architectural problems early in the development cycle. [Sch2002b] REACT is one of the first projects to successfully develop and apply architecture-centric, aspect-oriented assessment techniques to Unified Model Language (UML) based model representations of large embedded space applications [Sch2002a], [Sch2003a], [Sch2003c], [Sch2004a]. REACT receives periodic architectural information from contractors supporting various space programs. Because many of the large space programs are developing their architectural information using UML, REACT develops techniques to extract the architectural information from UML, and represent it using extensible markup language (XML) to support static and dynamic assessments. The architectural information REACT receives consists of a number of artifacts including requirement information (e.g. requirement specifications mapped into behavioral use cases that describe courses of action (COAs) for the system to support), interface control documents (ICDs), and architectural models (e.g. UML), test procedures, and code. The formats of the architectural information vary. It is not uncommon to receive UML descriptions, spreadsheets, presentation files, text files, and other proprietary data formats. REACT's collection of tools parses these artifacts into analyzable representations. These representations are typically XML files. Over time, REACT began to accrue a collection of XML schemas to capture architectural information and requirement information that is not represented natively as UML. SARAEE was formed to study REACT's various representation schemas and develop a meta-schema strategy that would organize not only requirements and UML-architectural information, but also information related to testing and simulation analysis, and REACT findings.

Our experience with REACT revealed that frequently there are problem areas among the relationships of the various architectural artifacts. Figure 1 illustrates the typical relationships among the various program artifacts and identifies some potential problem areas. These problems may result in an implementation that does not meet requirements and testing that may be incomplete.

REACT's experience suggests that many of these shortfalls continue to persist because of the evolving nature of complex real-time embedded development [Sch2003a], [Sch2003b]. Supporting timely architectural assessment in this environment requires automated support tools to prepare the information for analysis and assist in its evaluation. REACT's core-technologies support aspect-oriented architectural assessment [Sch2002a]. How these REACT-developed techniques can be applied to support architectural assessment and representation of evolving architectures is described in [Sch2004b].

This report discusses three auxiliary support tools that were designed and developed by SARAEE and which have been applied by the REACT facility to support architectural analysis of several large space satellite systems. The tools include a requirement visualization tool that assists in the visualization of requirement dependencies, a tool to support REACT's aspect-oriented architectural assessment approach, and an early prototype tool that assists in investigating the potential impact of unanticipated concerns. In section 4, we describe our future plans.

REQUIREMENT VISUALIZATION

Contractor-provided use cases are typically divided into a large number of individual files often developed by several developers in parallel. Ensuring that the use cases capture a functionally complete and consistent view of all requirements is increasingly difficult as space-based requirements become more complex. Use cases are frequently grouped into related functional behavior called courses of action (COAs). The COAs are comprised of steps which are illustrated in Figure 2.

Different COA steps may invoke other COAs to support alternative COAs or exception conditions. COAs therefore can be functionally related or dependent on other COAs. This organization facilitates modularizing requirement dependencies. For example, invoking a common error logging "utility" course of action similar to a subroutine usage may satisfy functional requirements that need to perform an error logging service as part of alternative processing.

To support REACT's use case analysis, REACT developed tools to parse the use cases (typically provided as text documents) into a collection of XML files. When parsing use cases, it is possible to use other meta-information to filter requirements of interest. For example, require-

Figure 1. Relationships Among Architectural Artifacts

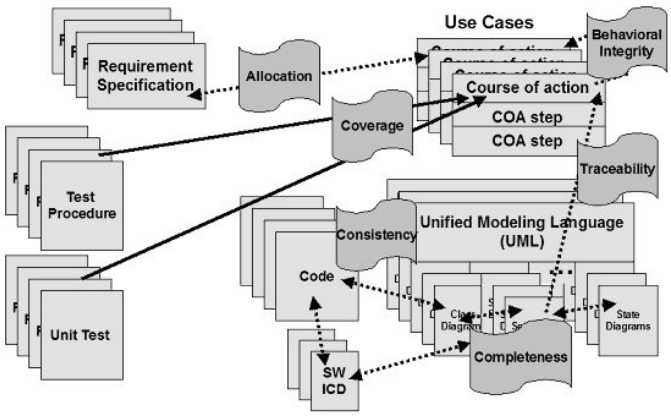


Figure 2. Course of Action Organization

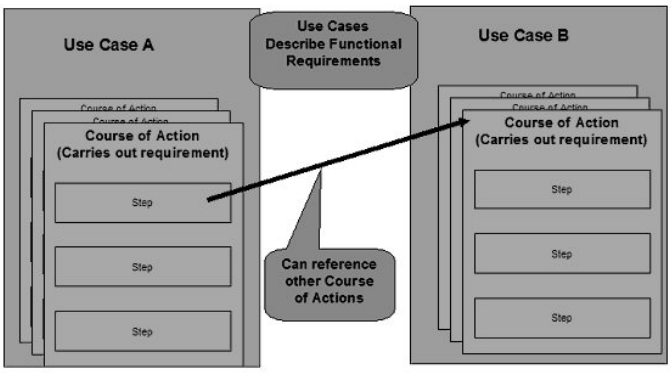
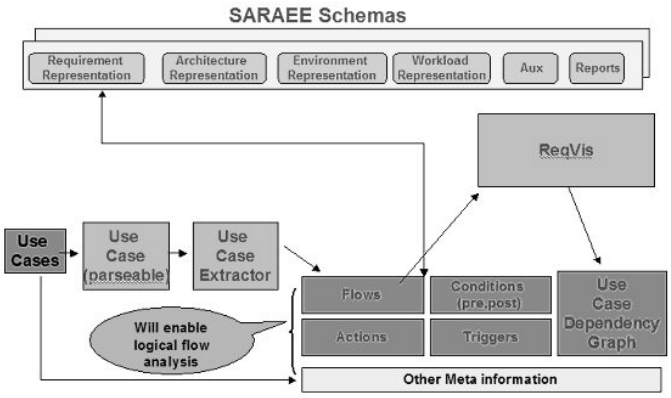


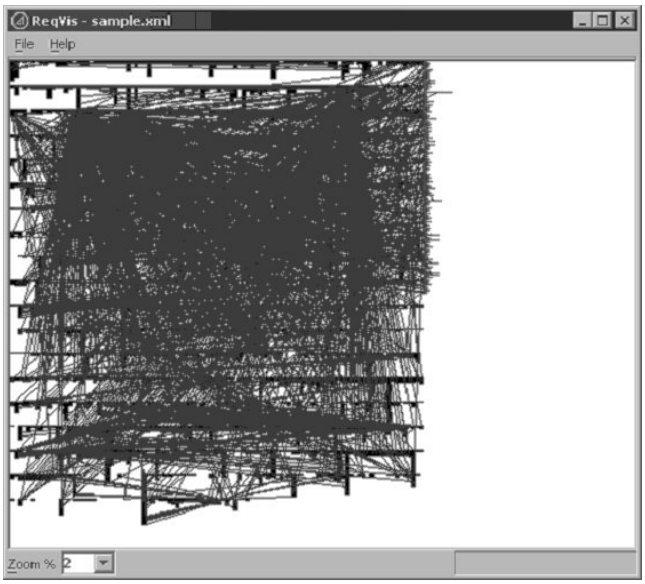
Figure 3. Use Case Analysis Process



ments could be pre-filtered based on the subsystem to which they are allocated, the build spiral in which they are implemented, or the test method. The generated XML files capture information that identify the flow of COA steps, the pre and post conditions, trigger, and specified requirement actions. The schemas for these files formed the basis of SARAEE's requirement representation schema. The requirement visualization tool, called ReqVis, was developed to support use case analysis of the generated XML files. This process is illustrated in Figure 3.

ReqVis is a tool for constructing a visual representation of use case requirements and their relationships as a graph of nodes and edges. It is not uncommon for an advanced satellite or ground system to contain several thousand individual steps to be carried out under various conditions making manual analysis extremely limited. The total number of steps (nodes) is one measure of a system's complexity, but it does not provide a complete picture. The number of links (edges) between courses of action steps represents the requirement interdependencies of the system, and a large number of interdependencies can affect our ability to understand the repercussions on development schedules and test plan relationships. Consider the example of the error logging mechanism mentioned previously: all courses of action that make calls to the error logger cannot be tested fully until the error logger itself has been written and tested. Such dependencies need to be identified early to ensure that they do not impact development schedules and lead to costly delays. ReqVis allows the user to identify use case dependencies visually by building a graph representing the courses of action, their steps, and the calls that link them together. Figure 4 illustrates complex use case dependencies for an actual satellite system. Individual courses of action are displayed as vertical stacks of boxes linked by short arrows, each box representing one step and indicating the flow of control between steps

Figure 4. Use Case Dependencies



within a course of action. The longer arrows indicate calls from one step to another. The orientation of the arrow indicates which step is making the call. It is easy to observe courses of action that are either the source or target of many calls. In this example, one such course of action sits at the extreme left of the third row. The lines radiating out from it indicate that it either calls or is called by dozens other courses of action. Another item of interest in this screenshot is the ragged strip of boxes along the right edge of the graph. These lines represent nonexistent courses of action that are nonetheless the target of one or more calls. These represent errors in the use case documents which may be caused by are missing use case alternatives, improperly specified requirement semantics, or subtle typos that create dangling links that have not been corrected. Detecting such discrepancies manually would be quite time consuming .

We have used this capability of selecting subsets of requirements to identify early a collection of subtle requirement allocation problems that could have affected a planned subsystem test. In one example, the contractor had scheduled individual subsystem tests. REACT pre-filtered requirements based on the requirement allocations provided by the contractor for the subsystem under test. These pre-filtered requirements were then analyzed using ReqVis to identify the "dangling" requirement references. Upon inspection, REACT found that some requirement references were missing, while others were valid but unallocated to the subsystem. In addition, REACT discovered that the subsystem under test depended upon the successful functional capability of another subsystem that was not scheduled for testing until much later. The discovery of unallocated requirements and subsystem dependencies enabled the contractor to revise their requirements, allocation, test plan strategy, test procedures, and schedule to correct these problems early.

These kinds of results have proven the usefulness of ReqVis, even at this early stage of development. Features that still remain to be added include filters for dynamically displaying only a particular subset of the courses of action and cycle-testing capability. The latter is particularly helpful, as it will be able to locate relationships of the form "A calls B, which calls C, which calls A" that could lead to processing dependency loops that make testing problematic. In the future, additional information (such as triggers, pre-conditions, and post-conditions) will be attached to each of the use cases, allowing them to be easily displayed and analyzed. When combined with the subset capability mentioned previously, the ability to locate and alter specific portions of the use case set could prove extremely powerful.

Figure 5. Aspect Processor Aspects Screen

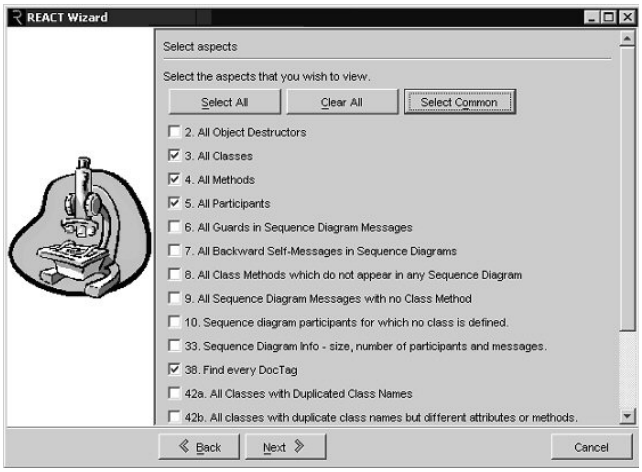
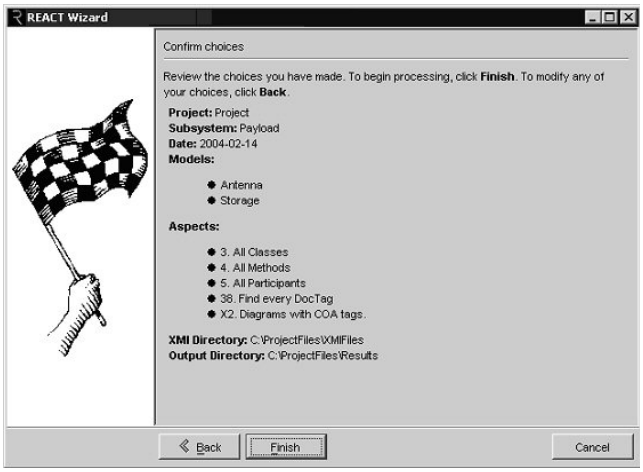


Figure 6. Aspect Processor Confirmation Screen



ASPECT PROCESSING

REACT's aspect-oriented approach to architectural assessment and how REACT results have supported UML analysis are discussed in [Sch2003a], [Sch2003c], [Sch2004a]. In this section, we describe how an Aspect Processor tool, shown in Figure 9, was created to effectively manage the growing volume of contractor data and efficiently perform UML-based architectural assessments. Although REACT developed several aspect-oriented assessments to investigate architectural consistency and completeness concerns, the processing needed to perform the assessments requires a complicated manual intervention of several application tools. To support early discovery of potential problems, REACT began to acquire large collections of contractor data. With the proliferation of REACT tools and the frequency of data deliveries from contractors, the amount of data manipulation that had to be performed by hand became a bit overwhelming. The Aspect Processor provides a simplified, wizard-style interface to streamline the extraction and aspect analysis portion of the tool chain. After selecting the current program and known UML models of contractor data to analyze, the user is presented with a selection of aspects that can be applied over the contractor's pre-processed UML model. The aspect screen is shown in Figure 5.

The Aspects screen lists the currently available aspects and allows us to select the ones that we wish to run against the selected models. One useful additional feature The "Select Common" button selects a subset of the aspects that have been previously designated as the most applicable to the selected subsystem. This is convenient when running the same set of aspects against multiple models for a given project to collect trending information. The common aspects are numbered to facilitate selection. Having selected the aspects we're interested in, we specify the UML files (in an XMI format) to process and specify the directory for the output files. The XMI input files will always be copied into a hierarchical directory structure broken down along the same lines as the wizard: project first, followed by subsystem and then date. In certain cases (specifically, when we run either every available aspect or just the "common" aspects for the subsystem), a copy of the output will also be stored alongside the source files that were used to generate them. This caching mechanism allows us to simply return the generated output file immediately if the same parameters are passed to the wizard in the future. The directory selection is performed via a standard file browser window. Once the directories have been selected, we can proceed to the final summary of choices shown in Figure 6.

When the "Finish" button is clicked, the XMI input files will be copied to the appropriate location in the file system hierarchy (with the necessary directories being created, as required), the Extractor tool will be run on each model to convert it to REACT's internal representation format, and the Aspect Processor tool will run the selected aspects against the selected models. The output will then be placed in the selected output directory, along with a cached copy in the hierarchal directory structure if the conditions outlined previously were met.

The Aspect Processor tool eliminates the problem of shuffling contractor and intermediate data from directory to directory, avoids an error prone process of manually entering the necessary parameters for each individual run of the Extractor, dynamically creates the appropriate XML input that defines the selected aspects, executes the aspect evaluation, and copies the final output to a convenient location. Aspect processing reduces to an easy series of wizard windows, followed by a final processing step as the instructions are carried out. In addition, it enforces good file organization by storing the source and intermediate files in standardized locations, as well as caching useful results for later use. Through use of the Aspect Processor tool, the total time required to convert contractor data from XMI files to useful results have been reduced from hours to minutes.

In the future, the REACT's generation of assessment products will be expanded to include other REACT tools such as automating the conversion of UML models to XMI through integration with commercial UML tools, and automating the processing of the various Perl scripts that are used to perform use case analysis. Because the wizard is implemented using the Java programming language, it can easily be integrated into other tool frameworks such as Eclipse. For pre-processed REACT assessment products, we are planning to develop browser-based interfaces to disseminate REACT findings to members of program offices via the Web. This would be a significant step in improving the availability of assessment products and in lowering the threshold for end users to generate and obtain REACT reports directly, without having to request the help of REACT team members.

IMPACT ASSESSMENT SUPPORT

During the design and development of complex space satellite systems, system engineering may require unforeseen changes. Significant changes often arise in the form of an engineering change proposal (ECP), and the sponsoring program office frequently requests what the impact of the change would be on the evolving architectural artifacts of the system. Figure 7 illustrates how the impact of a single concern can cut across many architectural artifacts.

In practice, a typical change will create one or more concerns that cut-across many representation/information artifacts, often in a highly project-specific manner. As a system evolves, not all concerns of interest are known a priori, and therefore it is unreasonable to expect that all aspects will be explicitly managed with explicit representations

Figure 7. Cross-Cutting Concern

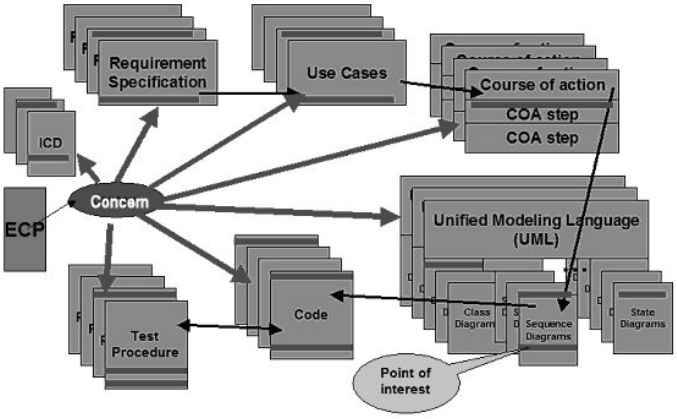


Figure 8. Impact Assessment Results

```
<?xml version="1.0"?>
<RModelAspects>
    <SourceFileSet>
        <SourceFile name='Project_2002_09_14_Subsystem1_react.xml'/>
        <SourceFile name='Project_2002_09_14_Subsystem2_react.xml'/>
        <SourceFile name='Project_2002_09_14_Subsystem3_react.xml'/>
        <SourceFile name='Project_2002_09_14_Subsystem4_react.xml'/>
    </SourceFileSet>
    <ResultFile name='out.xml'/>
    <AspectSet>
        <Aspect name='2. String Search'>
            <Query name='//Coperation//vx'>
                <QAction filter='Package Class Coperation' name='printpath'>
                    <Result>
                        <ModelResult length='4' source='Project_2002_09_14_Subsystem1_react.xml'>
                            <Path>Os_augs:vxWorks_msgQ:msgQCreate</Path>
                            <Path>Os_augs:vxWorks_msgQ:msgQSend</Path>
                            <Path>Os_augs:vxWorks_msgQ:msgQReceive</Path>
                            <Path>Os_augs:vxWorks_msgQ:msgQNumMsgs</Path>
                        </ModelResult>
                        <ModelResult length='5' source='Project_2002_09_14_Subsystem2_react.xml'>
                            <Path>Os_augs:vxWorks_semaphore:semBCreate</Path>
                            <Path>Os_augs:vxWorks_semaphore:semCCreate</Path>
                            <Path>Os_augs:vxWorks_semaphore:semMCreate</Path>
                            <Path>Os_augs:vxWorks_semaphore:semGive</Path>
                            <Path>Os_augs:vxWorks_semaphore:semTake</Path>
                            <Path>Os_augs:vxWorks_semaphore:semFlush</Path>
                        </ModelResult>
                        <ModelResult length='3' source='Project_2002_09_14_Subsystem3_react.xml'>
                            <Path>Os_augs:vxWorks_task:taskSpawn</Path>
                            <Path>Os_augs:vxWorks_task:taskDelay</Path>
                            <Path>Os_augs:vxWorks_task:taskIdVerify</Path>
                        </ModelResult>
                        <ModelResult length='3' source='Project_2002_09_14_Subsystem4_react.xml'>
                            <Path>Os_augs:vxWorks_watchdog:wdCreate</Path>
                            <Path>Os_augs:vxWorks_watchdog:wdStart</Path>
                            <Path>Os_augs:vxWorks_watchdog:wdCancel</Path>
                        </ModelResult>
                    </Result>
                </QAction>
            </Query>
        </Aspect>
    </AspectSet>
</RModelAspects>
```

within the architectural artifacts. REACT found that UML representations were often a combination of different levels of abstraction making it difficult to discern abstract conceptual model usage from implementation-oriented model usage [Sch2003a], [Sch2003c]. Additionally other architectural problems with inconsistencies, behavioral incompleteness, and weak traceability from requirements, through design and code, tended to make impact assessment of requirement and design changes difficult. Given these architectural handicaps, SARAEE studied how REACT's aspect-oriented assessment techniques could be applied to assess the impact of changes to an evolving architecture [Sch2004b]. The Impact Assessment tool is a proof of concept tool that demonstrates the feasibility of applying REACT's aspect-oriented assessment techniques to an evolving architectural description. The approach taken was as follows:

1. A concern of interest would be identified.
2. From that concern, a means to specify an architectural "handle" that characterizes that concern would be identified. Identifying an architectural handle usually requires recognition of program-specific UML usages for a given architectural description. For complex concerns, many "handles" might be necessary to achieve a full characterization. In particular, architectural augmentation aspects may be needed to clarify this characterization as the architectural representations evolve.
3. Finally an aspect would be written to exploit the architectural handle(s) to identify points of interest related to the concern, and if necessary take appropriate action.

The Impact Assessment tool was used in the following example to demonstrate this approach in a straightforward manner.

1. The concern of interest was: Identify commercial tool dependencies. In our embedded example, this concern meant identifying places within the architecture where a commercial operating system (e.g., VxWorks) was being used.
2. In this example, the contractor had pre-fixed method invocations with a Vx substring. Finding all methods of all classes with this prefix would identify places of VxWorks dependencies. The Impact Assessment tool provided a means to specify an arbitrary text string to search over all architectural methods.
3. Although it is possible to develop augmentation aspects that could annotate or tag points of interest based upon any collection of architectural handles, the proof of concept example did not require this. In this example, the aspect to search over all architectural methods only required the identification of methods. Figure 15 illustrates the XML data set generated by the Impact Assessment tool for the VxWorks query.

Prior to the Impact Assessment tool, REACT used fixed, pre-defined aspects to search and collect specific architectural information. The Impact Assessment tool provided a rudimentary capability to specify a variable string argument to specialize an architectural search. The REACT team applied this concept to an actual engineering change. The change arose because the specifications for an external device with which the contractor was to interface had changed. The program office was interested in all use case requirements, design artifacts, and code that could be affected by the change. Using the Impact Assessment tool, string identifiers characterized the concern. With this characterization, REACT not only discovered all the requirements and design artifacts of interest, but also found requirement allocation and traceability problems that prevented a direct identification to the implemented code.

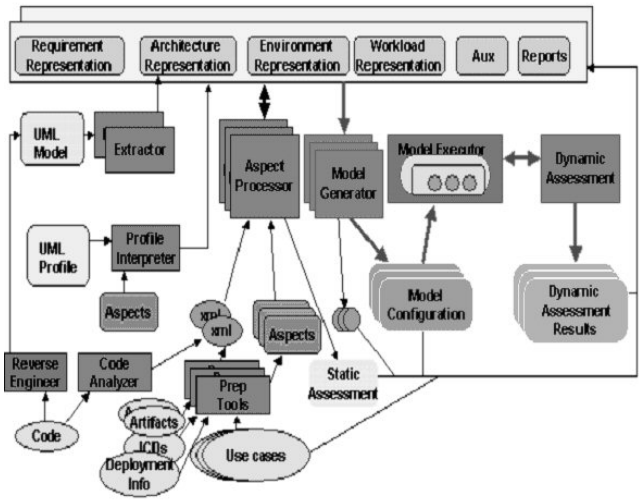
By reverse engineering implemented code into UML, REACT was able to identify several areas, impacted by the engineering change, where the reverse engineered design did not match the design specification. These discoveries proved useful, not only in clarifying the requirement dependencies on the external device, but also in correcting design disconnects within the implementation.

Currently the impact assessment tool takes an identified string and finds the points of interest. We plan to extend the impact assessment tool by providing the user the capability to create aspects that can not only find requested points of interest but also take some functional action to augment the representation. This capability can be quite powerful to configure the model generator to support dynamic assessment. Future work on the impact assessment tool requires developing the architectural schemas to support more sophisticated augmentation of meta-information.

CONCLUSIONS AND FUTURE WORK

The architectural evaluation support tools developed under SARAEE have been successfully applied by REACT to identify and resolve real architectural issues of embedded space satellite systems. Currently SARAEE is developing various schema representations to broaden the

Figure 9. Aspect-Oriented Architectural Assessment



scope of impact assessment. Figure 9 illustrates SARAEE's schema areas under development and the underlying tools to support their analysis.

We are currently investigating ways to improve the usability of these tools. One promising approach is to integrate REACT's various assessment capabilities into an Eclipse-based environment [Ecl2004] as a collection of plugins. We also plan to migrate REACT's architectural representation schemas into the UML 2.0 meta-model and develop both domain-independent and domain-dependent profiles to support more sophisticated dynamic assessments. One benefit from the Eclipse plugin approach has been the ability to support aspect composition. Our ongoing efforts continue to evolve the tools to meet the changing demands of various program offices.

REFERENCES

[Ecl2004] http://www.eclipse.org

[Sch2002a] Schmidt, P., Duvall, R., Lankford, J., Mulert, G., "Evaluation of Aspects in UML Models," *Proceedings of the 2002 Ground Systems Architectures Workshop*, March 13-15, 2002, El Segundo, CA.

[Sch2002b] Schmidt, P., Milstein, J., Duvall, R., Lankford, J., Rivera, J., "Lessons Learned Using REACT: An Architectural Testbed for Real-time Embedded Systems, in *Proceedings of the 2002 Information Resources Management Association International Conference*, Seattle, WA, May 19-22, 2002.

[Sch2003a] Schmidt, P., P., Alvarado, S., Milstein, J., Mulert, G., Duvall, R., Rivera, J., "A Systems Engineering Perspective of Aspect-oriented Software Architectural Analysis using UML," in *2003 Aspect-Oriented Software Development Workshop on UML*, Boston, MA, March 18, 2003.

[Sch2003b] Schmidt, P, Alvarado, S., Rivera, J., Milstein, J., "Architectural-Centric Representation for Design Diversity and Program Evolution," in *2003 Proceedings of the Ground Systems Architectures Workshop*, March 4-6, 2003, The Aerospace Corporation, El Segundo, CA 90245. See http://sunset.usc.edu/gsaw

[Sch2003c] Schmidt, P., Duvall, R., Mulert, G., Milstein, J., Rivera, J., "Aspect-Oriented Architectural Analysis using Multi-level Modeling of Complex Systems, in *Proceedings of 2003 International Resource Management Conference* , Philadelphia, PA, May 2003.

[Sch2004a] Schmidt, P. Milstein, J., Alvarado, S., "An Analysis of Aerospace Software Architectures Using Aspect-Oriented Programming Principles," Aerospace Technical Report ATR-2004(8343)-01, 28 May 2004.

[Sch2004b] Schmidt, P. Milstein, "Representing and Evaluating Software-Intensive Architectures that Evolve," Aerospace Technical Report ATR-2004(8343)-03, 31 Aug 2004.