

Managing Evolving Product Line Architectures

Ménage is an environment for managing evolving product line architectures. Ménage builds upon our existing representation for software architecture, namely xADL 2.0, to provide three capabilities that are specifically geared towards managing an evolving product line architecture.

1. Ménage supports the definition of a product line architecture as a set of core architectural elements that is extended with variation points. Variation points are the optional and variant elements with which individual product architectures can be distinguished from each other.
2. Ménage explicitly tracks the evolution of all individual architectural elements as well as the overall product line architecture with a versioning mechanism that automatically creates a history of all changes.
3. Ménage provides integrated architectural differencing and merging algorithms that allow a developer to precisely understand the differences between two product architectures and automatically propagate these differences to a third product architecture.

Variability

Figure 1 shows the graphical user interface provided by Ménage. The panels on the left side list component, connector, and interface types that have previously been defined and can be used in the construction of new types. In this particular instance, a new type is being defined that represents a Word Processor product line architecture. The Word Processor standardly consists of a user interface component of type VisualBasic, a layout engine component of type FastLayout, and a storage component of type FileSystemStorage, all connected with a couple of connectors (Bus1 and Bus2). The two other components represent variation points and are discussed below.

When an optional component, connector, or interface is added to a product architecture, an architect is requested to provide a Boolean expression that will serve as the guard for that optional component. Once the guard is specified, the element is added and highlighted with a dashed border to indicate its optional status.

In the example of Figure 1, the print component of type HPPrint is optional. Its guard can be viewed and edited simply by right clicking on the component. Note that, because the print component is optional, the link to the connector Bus1 is automatically optional as well: if the print component is included in a particular product architecture, the link is included as well, otherwise, it is left out.

A variant component or connector is added in exactly the same manner as a normal component or connector: the architect simply chooses a desired component or connector type from the panels on the left. This translucent behavior is possible since variability in Ménage resides within special variant types that, except for their constituents, are treated in exactly the same way as ordinary types. Whereas types generally consist of arbitrary sets of interconnected components and connectors, constituents of variant types are limited to other types that: (a) are either all component types or all connector types, (b) are not connected to each other, (c) each have a mutually exclusive Boolean guard, and (d) all exhibit at least the same set of interfaces as the variant type. These four restrictions guarantee compositionality within the remainder of the product line architecture when one of the constituent types is selected to be incorporated in a product architecture.

Whenever a new optional or variant element is added to a product line architecture, the presence of a new variation point has the effect of creating one or more new product architectures. Selection of one of those product architectures by hand is automated by Ménage. Given a set of desired selection constraints, Ménage reduces the full product line architecture to a product line architecture that contains only those product architectures that satisfy the selection constraints. Often, this will result in a single product architecture, but sometimes it may have the effect of creating one or more new product architectures. Selection of one of those product architectures by hand is automated by Ménage. Given a set of desired selection constraints, Ménage reduces the full product line architecture to a product line architecture that contains only those product architectures that satisfy the selection constraints. Often, this will result in a single product architecture,

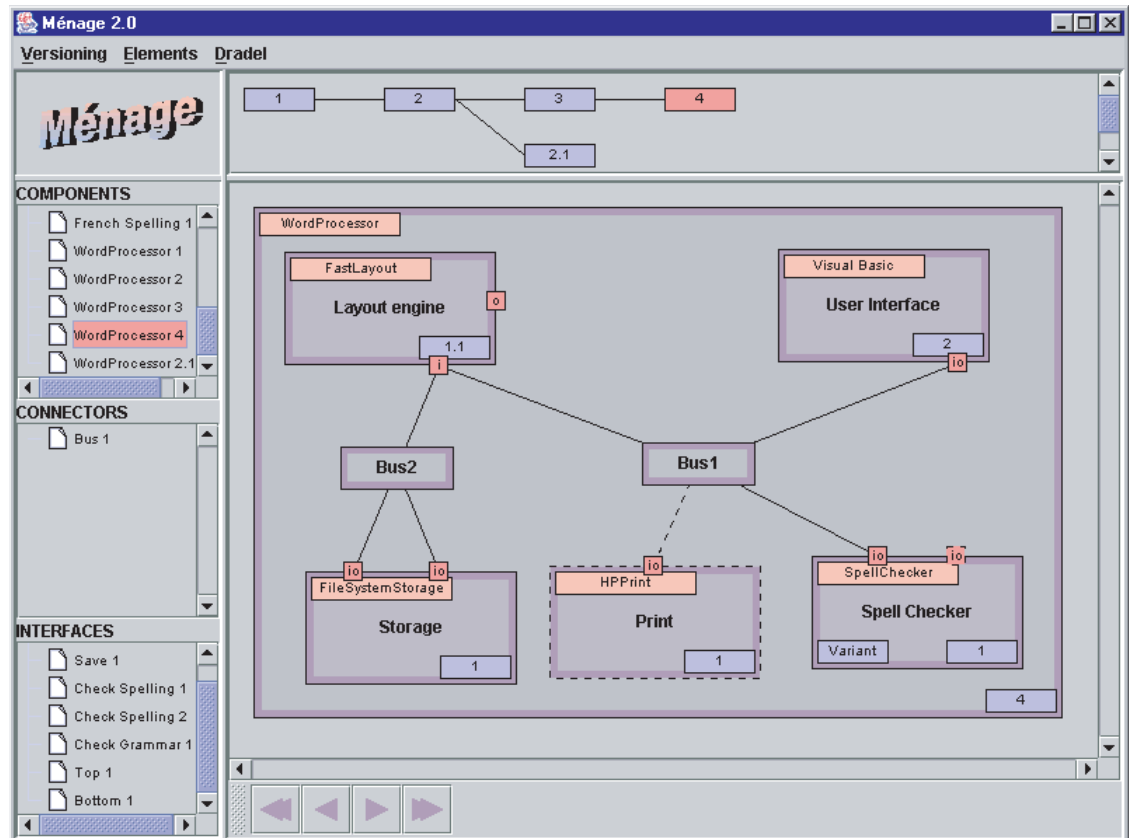


Figure 1. Screen shot of Ménage GUI

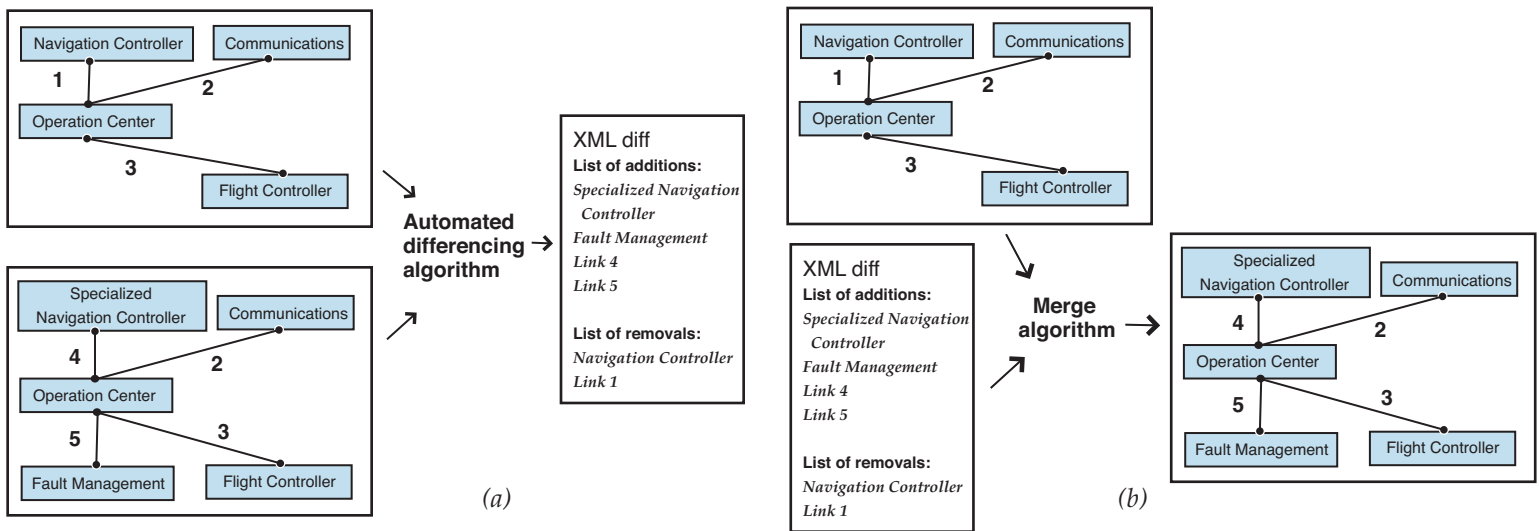


Figure 2. An automated differencing algorithm creates a xADL 2.0 XML architectural patch describing the differences between two product architectures (a), and an automated merging algorithm allows incorporation of such a patch in a third product architecture (b).

but Ménége also allows the result to be a "mini" product line architecture that only contains a subset of the product architectures. Selection of an actual product architecture out of the "mini" product line architecture can then be deferred to a later stage in the development process.

Evolution

A critical part of Ménége's ability to manage the evolution of a product line architecture is provided by its integrated versioning facility. Before any changes can be made to a product line architecture, Ménége requires an architect to check out the set of architectural elements they will be modifying. After that, the architect is free to manipulate those elements in order to change the product line architecture as desired by adding new product architectures, updating existing product architectures, and removing old product architectures. Once all desired changes have been made, the architect checks in the modified parts of the product line architecture. Ménége then automatically creates a new version of the product line architecture and, in the process, a history of changes.

Figure 1 illustrates this process in action. The top panel shows the version tree that resulted from a series of changes to the Word Processor product line architecture. The version that is highlighted, version 4, is the version of the Word Processor product line architecture that is currently being manipulated in the main panel. Old versions can be revisited simply by clicking on the version number that represents the version to be examined.

Architectural Differencing and Merging

While in theory the functionality as described above is sufficient to manage the evolution of a product line architecture, in practice, the job of an architect can still be rather cumbersome. First, if a product line architecture consists of many different product architectures, it becomes increasingly difficult for an architect to easily and quickly understand the differences among those different product architectures. Second, if an architect makes a change to one product

architecture that may be beneficial to other product architectures, the above functionality only allows the architect to propagate the changes by hand and make them one-by-one in each of the relevant product architectures. Clearly, both of these issues must be addressed by any kind of environment aspiring to be effective at supporting the management of evolving product line architectures.

Ménége provides automated support that addresses both issues. Specifically, Ménége provides an architect support for understanding differences between product architectures via an automated differencing algorithm and support for propagating changes from one product architecture to another in the form of an automated merging algorithm.

Conclusion

Ménége is an environment for managing the evolution of product line architectures. Ménége is unique in providing an architect with an environment that not only allows them to specify a product line architecture once, but also allows them to evolve this product line architecture as new product architectures are added, existing product architectures are changed, and obsolete product architectures are removed—all the while keeping a history of changes such that old versions of the product line architecture can be revisited and resurrected. Ménége supports an architect in performing these tasks by supporting three key functionalities: (1) the definition and recording of variation points, (2) the creation of historical versions based upon a simple check out/check in mechanism, and (3) the propagation of changes from one product architecture to another via the use of automated differencing and merging algorithms.

References

- [1] E.M. Dashofy, A. van der Hoek, and R.N. Taylor. *An Infrastructure for the Rapid Development of XML-Based Architecture Description Languages*. 24th International Conference on Software Engineering, 2002.
- [2] C. Van der Westhuizen and A. van der Hoek. *Understanding and Propagating Architectural Changes*. Working IFIP Conference on Software Architecture, 2002.

Contact Information

Professor André van der Hoek
Institute for Software Research
University of California, Irvine
Irvine, California 92697-3425

andre@ics.uci.edu
949-824- 6326
949-824-1715 (fax)

Information about Ménége, as well as links
to other software tools:

<http://www.ics.uci.edu/~egelink/menage/>
<http://www.ics.uci.edu/~andre/research/projects.html>

This material is based upon work sponsored by the Defense Advanced Research Projects Agency, and Rome Laboratory, Air Force Materiel Command, USAF, under contract numbers F30602-00-2-0607 and F30603-00-2-0599. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.