Zero-latency Engineering™ for Control Design

Dr.-Ing. Johannes Ernst, jernst@aviatis.com
Scott Washburn, swashburn@aviatis.com
Aviatis Corp. – http://www.aviatis.com/

1. Problem and Context

According to a common prediction (such as by Alan Kay), we are about to enter the "third paradigm" in computing. This new paradigm is dominated by computing devices that "recede into the background of our lives", in contrast to the previous ones which were dominated by very visible mainframes and PCs.

Although there is no consensus yet as to their appearance or functionality, it is a safe assumption that these devices will have some of the following characteristics: Computing functionality will be "built-into" some other device whose primary purpose does not appear to be computing. These products will function autonomously to some degree, interact with the "real world" through sensors and actors, have some kind of "user interface", and many of them will network with other devices. Because virtually all interactions between computing devices and the "real world" fall into the "Control Engineering" discipline, in effect, most 3rd wave computing devices will be control devices of some kind.

While some similarities exist, it is obvious that developing 3rd-paradigm products will be very different from developing most previous generations of systems. The most obvious difference is that almost all 3rd-paradigm devices will be mass-market products, in an intensely competitive, innovative, high-stakes global market. In contrast, many control systems today are produced in much smaller lots. It is probably also correct to predict that the computing capabilities in this new class of products will have price points for the end user that are in the same ratio to the cost of the previous paradigm's products (PCs) as the ratio was between PCs and mainframes.

Consequently, time-to-market and cost issues (which include marginal unit production cost, development cost as well as maintenance and service cost – some of which might well have to go down to zero for many types of products) will become vastly more important that they even are today.

As 3rd-paradigm products will not be "computing products", but products that serve some other purpose and just happen to include substantial computing power, these products will necessarily encompass components built with many different technologies, most importantly software, electronics hardware and mechanical hardware. The problem of designing those products, then becomes one of designing a multi-technology product, rather than one that is primarily focused on one type of technology (such as electronics, or mechanics, or software) as it is the state of the art today in many industries.

As a side effect, team-based, distributed product development will become even more important. While it is (theoretically) possible that one software engineer, for example, develops all of the software in a software product, very few, if any, people have the breadth of knowledge to develop and optimize a product that is built from many different technologies such as software, electronics and mechanical components. The same is valid for most teams. As this breadth of knowledge and experience is seldom found in location either, subcontracting and outsourcing will be even more prevalent than it is today.

The confluence of much stronger time-to-market and cost pressures with the requirement to develop and optimize multi-technology products and the associated need for increased outsourcing puts intense pressure on both the development methodologies for 3rd-paradigm products and the supporting software.

Unfortunately, the established engineering methodologies and supporting productivity tools today are generally ill-suited to face the challenges created by the design of 3rd-paradigm computing devices. This is most apparent in the following issues:

- the vast majority of engineering productivity tools today focus on improving the productivity of individual engineers, rather than the teams that engineers need to work in (which they do in virtually

all "commercially relevant" development projects). As every veteran project manager can attest (and that is compounded when designing 3rd-paradigm products) it is rarely a lack of individual productivity that keeps a project from being successful; it is much more likely that the team lacks productivity as a team. Some of the issues that this includes are lack of communication and collaboration, "no one told me" and "all the pieces work, but that system doesn't".

- all engineering tools today focus on one, or few, issues of the overall development process of 3rd-paradigm products, while ignoring the (much larger) rest of the development cycle. For example, there are excellent software, or hardware, or mechanical, or control design tools, each of which can be a great help in its domain, but each of which also ignores the respective other domains. This is understandable from the perspective of the tool vendor, even from the perspective of a single tool user. However, if the goal is end-to-end product development of multi-disciplinary products in the shortest possible time, with a minimum amount of risk and rework, an as-seamless-as-possible distributed, multi-disciplinary environment is needed that encompasses all the tool functionality that the development team needs.

Numerous approaches have been explored over the years in both industry and academia to alleviate the above problems, and often quite interesting work has been performed on many aspects of the problem. However, if we look at the broad commercial availability of ready-to-use software solutions to support integrated, multi-disciplinary development, none has been successful (and that includes several implementations that some of the authors have spearheaded and/or involved with!). While that might have been only inconvenient in the past, we have our doubts whether many of the 3rd-paradigm systems envisioned by industry pundits will ever be brought to the market without this development problem having been addressed in an industrial-strength way.

One can only speculate why these types of integrated software offerings do not exist, but in our experience, a combination of the strict separation of engineering disciplines (e.g. education, the methods and tools market) and the economics of having to develop N^2 interfaces has probably been the main culprit.

2. Approach

2.1 Overview

In order to address the above problem, we have developed a novel, internet-based approach to increase the productivity of multi-disciplinary, distributed engineering teams that use today's commercial vendors' best-of-class development tools in order to develop innovative computing-enabled devices under strong time-to-market and cost pressures. This approach is a combination of methodology and commercial, internet-based software technology that interacts with and integrates with today's users' best-of-class commercial tools in a collaborative, multi-disciplinary environment. We believe this approach elegantly sidesteps the technical and economic obstacles described above.

We call this approach Zero-latency Engineering™, where "latency" refers to unnecessary, time-consuming and high-risk problems caused by

- a lack of efficient communication and collaboration between all the (multi-disciplinary) members of a (distributed) development team, leading to high development risk, unnecessary complex and expensive products and long time-to market
- lack of sufficient tool integration so that multi-disciplinary development processes are possible.

Our approach has two primary components: a methodology and internet-based software that supports the methodology, both of which are being discussed below. As we fundamentally concentrate on the "systems" issues, and not the component issues, the methodology does not deal with "component design" (which is the design of components in one particular technology, such as electronics, software and/or hardware) and instead "delegates" them to the best-of-class component development methodologies that developers use today. Similarly, our technology supporting the methodology integrates and makes collaboratively usable the best-of-class tools that developers use today without competing with them.

This reflects the fact that "disruptive" new approaches, i.e. approaches that require development teams to substantially change the way they work and to change the software they work with, are very unlikely to work in industrial practice, which is the domain where 3rd-paradigm systems will need to come from.

2.2 Methodology

If one examines multi-disciplinary development processes as they are performed even today in some industries (such as in automotive, aerospace and some others), in most cases all development work originates from a common understanding of the "architecture" of the overall system to be built. This "architecture", or high-level blueprint, is often created and maintained by senior members of the development team, sometimes called "architects" or "systems engineers". The means to capture the "architecture" in many cases is some type of block diagram, whose primary concepts are components (sub-modules, etc.) and the information and/or material exchange between them. Often, these components are directly associated with different technologies and/or development teams, so the "architecture" model both defines technology and organizational boundaries.

Based on that high-level architecture, development deliverables are then assigned to more specialized development teams, most of which work with one particular technology (only). For example, one or more teams may develop software, others analog or digital electronics, others various mechanical components, sensors, actuators etc. each of which forms one or more of the components identified earlier. Note that the case that "reusable" components exist already – such as pre-manufactured parts or software libraries – is included here as a special case with component development effort zero.

In virtually all cases, each of the development teams for the various components will use their own tool chain – after all, they work with different technology (such as mechanical, electronics and software) and therefore the requirements for and the features of the used tools are necessarily different.

The fundamental obstacle to reliable and efficient development of multi-disciplinary systems is that:

- the heterogeneity of the problem requires the use of N tool chains
- that nevertheless need to be used in harmony so that the components developed using one tool chain are integrated into and compatible with the overall system – even more, that the overall system is optimized across all the different components and their domain (that includes hardware/software/mechanical trade-offs)
- in a sufficiently efficient manner so that time-to-market is not longer than necessary.

In other words, it is necessary to:

- drive the development of the multi-disciplinary system from an integrated understanding of all the artifacts developed during component development in the context, and "balanced" (to use this structured analysis term) with the high-level architecture model.
- that also includes other artifacts, such as documentation and its incorporation "by reference" of design artifacts from more specialized engineering tools, tests, user manuals etc.

Assuming the existence of such a live-linked, integrated representation of the overall system at all steps during the design (for the implementation see below) a fully integrated "holistic" systems development methodology is possible.

From the perspective of control systems design, this integrated view of the system under design has always been very necessary but difficult to provide in a consistent and reliable manner. In order to perform closed-loop analysis of a system, it is generally necessary to

- have access to an "context diagram" that includes the system under design and its anticipated environment (the "controller" and the "plant")
- an integrated model to represents the "controller" (which necessarily has to include sub-models for all the sub-components, ideally always consistent with the current design of that component in its respective technology) in an integrated manner with consistent and "connected" interfaces.

The existence of such an integrated systems model is a "side effect" of the methodology for multi-disciplinary systems development that we propose. As this integrated model is analyzed and changes are necessary to achieve or improve certain performance measures, it is always clear which components must be affected, or between which a trade-off may exist (something that is very difficult to do today, as they generally are disassociated from the underlying implementation technologies).

[Note: performance issues with respect to abstracting simulation models are outside the discussion in this paper.]

2.3 Software

The methodology discussed above is supported in a commercial implementation by Aviatis Corp. In order to provide maximum accessibility and ease of use, this implementation is entirely web-based, provided as a subscription service and accessible through a commercial web browser and the Aviatis website at http://www.aviatis.com.

After users have authenticated themselves to the Aviatis website and identified a project that they would like to work on, a screen like the one shown in Figure 1 is presented to the user.

On the left hand side, an outline of the overall project under development is shown. This outline is user-configurable, typically project-specific and shared by all engineers on the development project.

When a user selects any item in the outline, the "content" of that item is shown in the pane on the right, following a common PC model.

Generally, the items referenced in the project outline are "live-imported" from the tools in any of the tool chains used by any participant in the (distributed) product development project. This is a major departure from approaches that predate ours: once a design artifact has been live-linked into the project outline, the design artifact is (conceptually) integrated into that project outline "by reference". In other words, as the development team continues to make changes to their component design/implementation using whatever tool they choose to use, these changes will be "pushed" into the artifact's representation within the context of the project outline.

In Figure 1, the artifact shown is a control systems model imported "live" from a commercial control systems modeling and analysis tool. Our "live-link" technology causes the diagram shown to be updated automatically (not requiring a reload or similar by the user) as soon as the underlying model in the control design tool is changed.

It is important to emphasize that all the artifacts displayed to the user in his browser are aware of their full semantics, what is linked is not just "text and pictures". For example, the Gain Block in Figure 1 is aware of the fact that it is a gain block (and not just a green rectangle) and thus context-specific operations can be applied directly from the browser. It is also important to mention that users do not need to have the underlying tools installed to examine models.

Once a (or more) users have linked their artifacts into the project's shared live project outline – a mouse-click operation in the browser – users can establish semantic relationships between their design artifacts and other design artifacts from other participants in the project, regardless of their location (could be a subcontractor), tool chain used (works across vendor boundaries) and even engineering disciplines (e.g. the control design model could be linked to mechanical or analog electronics models that "implement" the concepts in the model).

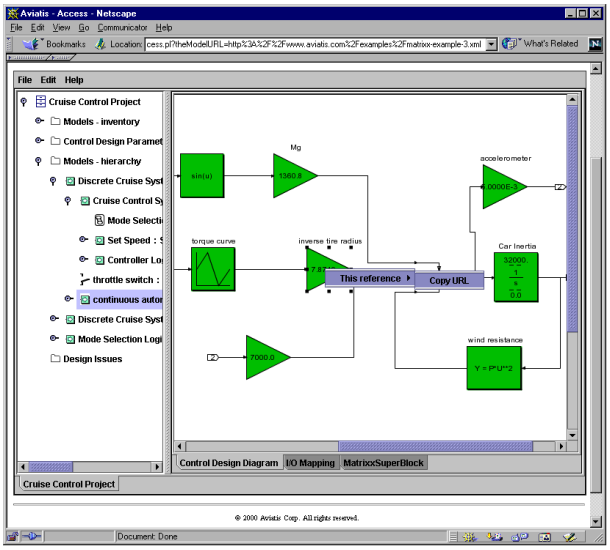

Figure 1: User interface example, here are control design model

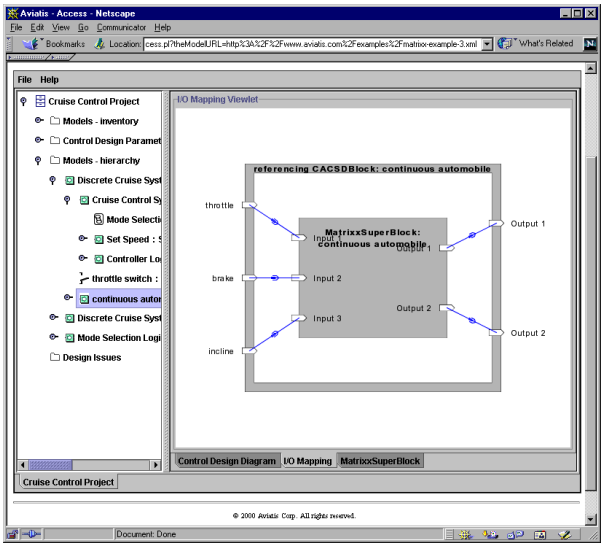

Figure 2: Connectivity between artifacts across boundaries of location and tool chain

An example for this linking is shown in Figure 2. In this diagram, the interface between a design artifact in one tool chain using one engineering discipline (e.g. a data flow diagram) and another artifact in another tool chain, potentially in a different location using employing potentially a different engineering discipline.

A summary overview over the deployment architecture is given in Figure 3: users access their design information through a Java applet in their browser (which they obtain by logging into http://www.aviatis.com/). This applet then connects to the Aviatis Integration Server which is typically deployed with a user organization's corporate firewall. The Aviatis Integration Server connects to the data stores of the tools that are used on the project across the network using a set of tool or format-specific "enablers" (which are software components that can translate a native data store format into the internal, fine-granular Aviatis representation). After a data source has been brought into a project for the first time, the enablers also "watch" changes to that data source in order to present always-up-to-date information to all users currently accessing a contained piece of information. The Integration Server also makes sure that relationships created by one user between elements from two different data sources are instantly "pushed' to all other users in order to avoid this major time killer, which is project participants working on outdated information.

All the information handled by Aviatis technology (through enablers, the Integration Server and the user interfaces) is modeled by an interdisciplinary meta-model. An overview of this meta-model is shown in Figure 4: all meta-objects are derived from a common root in the inheritance hierarchy. Several packages of meta-objects refine the meta-model, first into generic structuring concepts, then into domain-specific and finally into tool-specific models.

This meta-model is a format instance of a certain meta-meta-model (not further discussed here) and can act as a generic integration tool for domain-specific meta-models. Generally, we reuse well-established meta-models in the industry as much as possible (e.g. UML, CDIF, STEP etc.) and provide

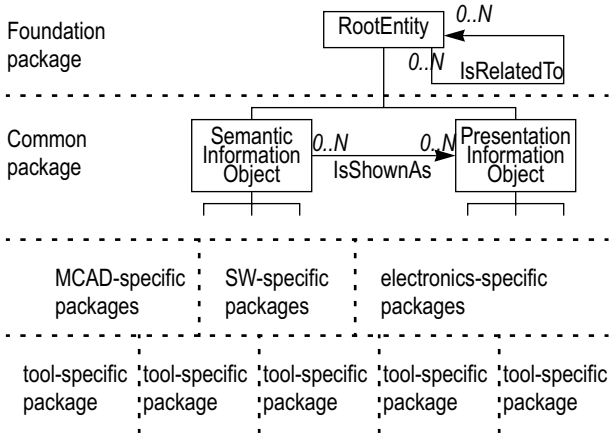

Figure 4: Meta-model overview

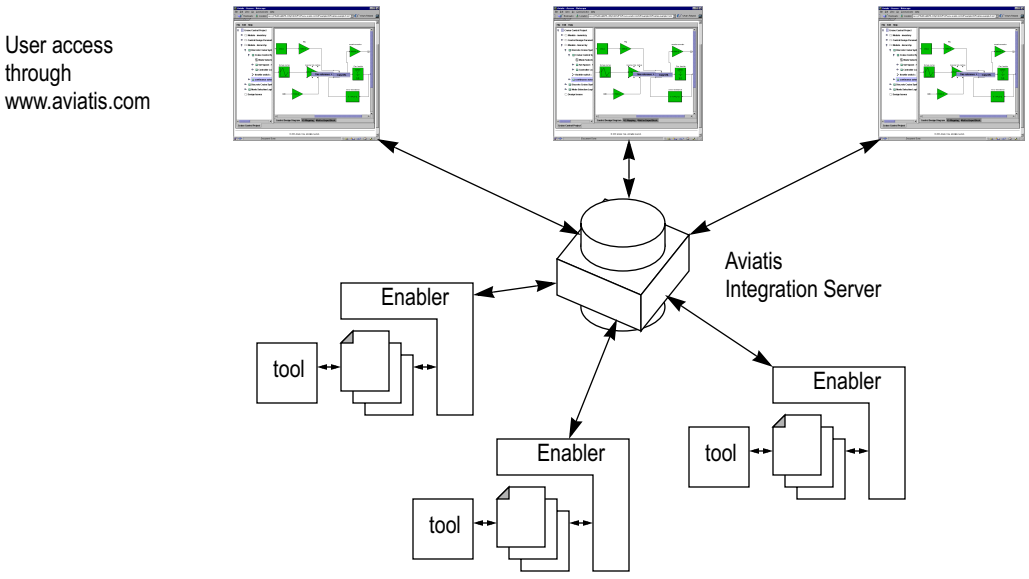

Figure 3: Architecture overview

tool-specific or proprietary extensions to achieve representational faith for tool information and enable cross-domain relationships.

2.4 Extensibility

Multi-disciplinary development inherently requires the use of not only large amounts of design information, but also very heterogeneous types of information. This is often most apparent in the sheer amount of tools used on a project, which, for certain types of complex systems, can easily number dozens. To compound the problem, not all tools tend to be available on the market, as engineering software developed in-house is common, and sometimes is more important for the success of a project than commercially available software.

Almost inevitably, any commercial offering by any vendor addressing a multi-disciplinary development process will miss support for some of the software packages employed. Thus, openness and extensibility of the implementation is very important for the success of this approach in order to enable 3rd parties to "fill the (unavoidable) gaps" and thus indispensable for large projects.

In our approach, there are three primary elements of the architecture that can be extended:

- the meta-model can be extended through subtyping (to cover additional modeling concepts) and through the creation of additional associations between the concepts provided by the (extended) meta-model (to cover additional relationships between the multi-disciplinary design artifacts)
- additional enablers can be developed that extend coverage of this approach to tools and data formats not yet covered. An enabler developer's kit is available that allows 3rd parties to rapidly develop functionality to convert information in additional file formats and/or repository structures so this information can be included in this project-wide project structure;
- 3rd parties can and are encouraged to use our developer's kit to implement new viewlets, i.e. user interface components that allow the user to visualize and/or interact with the integrated, multi-disciplinary, multi-tool model available through the browser in additional ways. Very little research has been done up to now on visualizing complex relationships between design artifacts from different engineering disciplines, and through the availability of a simple-to-use viewlet extensibility architecture, we hope to encourage not only this research process but also commercial-strength implementations of its results.

Obviously, research will not be translated into commercial-strength implementations unless it is possible for 3rd parties to generate a profitable revenue stream from their implementations. To enable the latter, Aviatis offers to redistribute extensions to our platform to our customers for a percentage of the revenue that the 3rd party generates with its implementation. Generally speaking, 3rd parties' extensions appear seamless within the Aviatis service.

3. Summary and work in progress

The Aviatis approach has been developed over a period of several years based on the authors' decade of experience with the design of heterogeneous embedded control systems. A commercial implementation with enablers for several commercial engineering tools and related software has recently been made available as a product through http://www.aviatis.com/. First practical experiences from pilot projects by blue-chip engineering organizations have been obtained, and several extensions by third parties are in progress. Due to its ease of use, minimal training requirements, extensibility and integration with an engineering organization's existing software infrastructure (rather than requiring users to change their infrastructure or way of working) we expect widespread commercial adoption by teams developing 3rd-paradigm computing systems.

The development of additional enablers is currently in progress, as are integrations with various "management information systems". Over time, we will add various features to address additional needs of very large, multi-organizational development teams. Together with our partners, we will also apply our technology to other domains where it is necessary or advantageous to relate, in a fine-granular manner, information contained in disparate information islands. We hope and expect that the widespread availability of this innovative, integrating "middleware" technology can and will bring the community's ability to develop next-generation products a major step forward, and specifically with respect to the economically viable design of the exciting 3rd generation computing products to come.