BUILDING TOOL SUITE FOR AADL

Jean-François Tilman
Axlog ingénierie
19-21 rue du 8 mai 1945
94110 Arcueil, France
(+33) 1 41 24 31 33
Jean-Francois.Tilman@axlog.fr

Abstract Architecture description languages (ADLs) are more and more considered in system engineering to model real-time applications (avionics, transportation, critical industrial systems, etc.). They provide means to formally specify architectures and support their design from the capture of the needs to the final validation. ASSERT, a european integrated project (IP) tackling improvement of system engineering process, is an illustration of this consideration.

Among all the existing ADLs, a few must be attentively considered because they should spread in the future. This is the case for AADL, an ADL initially dedicated to avionics applications, and now designed to support any domain critical application.

AADL may play a great role in industry to improve software and system development process. To achieve this objective, we need to strongly combine the description capabilities of AADL with tool suites used to develop, generate or test the system. This means that such tool suites have to explicitely support AADL.

AADL is based on MetaH, which is both an ADL and a tool set supporting it. In this paper, we will consider how an equivalent AADL tool suite could be built, possibly based on MetaH tools. We will also consider the industrial domains where it could be adopted and the role it could play.

1. Introduction

Architecture description languages (ADLs) are more and more popular in industry. After their development and use in laboratories, a lot of research is currently led by industry to transfer and integrate them into actual system engineering process. ADLs provide a means to formally describe system architectures. Many activities require such descriptions, and the introduction of such a formalism can help in their improvement.

The use of a description language in an industrial process requires its full support by the tools involved in this process. If not so, this language will not be adopted by users because its manipulation is unpracticable. Among all the existing ADLs, a few have the vocation to cover large parts of the industry needs. AADL [aadl] is one of them. For this ADL also, the support by tools is an issue to be considered.

In this article we will first consider the possible roles of ADLs in system engineering process and we will compare approaches of several domains, mainly automotive and avionics. Secondly, we will talk about the tool aspect of this subject: why is it important to support an ADL with tools, what are the existing bases to build such tools for AADL, what can these tools look like.

2. Interest of ADLs

Role

When developing computer-based systems, we must avoid inconsistencies in each phase of the development: the capture of the needs must be complete and consistent, the specification must cover all the needs, the design must fulfil the specifications, and so on, until the last phases of the development cycle and the validation.

What is difficult here is to ensure that (1) each piece of information in a given phase is completely taken into account in the next phase during the refinement and (2) that the contents of each phase are consistent. This problem deals with the management of information. It is more accurate during the first phases, typically where we often use natural languages to describe the needs and the specifications.

The best way to solve this problem is to use formalism, and particularly use a formal means to describe the system architecture and the related requirements. This is the role of architecture description languages (ADLs).

"ADL" is a very generic acronym, and many ADLs exist to describe architectures of various domains : mechanics, electronics, software, etc. We will only consider those dealing with computer-based systems, that is, systems containing computers embedding software. Among them, some are specialized for a particular application domain (avionics, automotive, etc.), or for particular aspects (synchronous approach, distribution problems). For our purpose, we are interested by ADLs supporting the whole system, not only a subset, and during all the development cycle.

Avionics domain

Avionics is a domain where industry has early considered the possible interest of formal descriptions in development process. These considerations are

motivated by the high quality level required in this domain. The following examples show activity of avionics industry in these questions.

In 1990, MetaH [metah] has been initiated by Amcom (US army) and Honeywell. MetaH is both an architecture description language and a tool set. It enables the description of avionics software architectures in a bottom-up approach and the automatic generation of what is needed to agregate the user components and execute them in a given target.

MetaH has demonstrated the interest of this technique in terms of cost and duration of developments. Some operations like retargeting a source code for a new hardware can be significantly improved (up to 70 %). However MetaH has never been largely deployed in industry.

In 2000, Amcom and Honeywell decided to create an international standard for an architecture description language based on MetaH. This new language is AADL (originally meaning Avionics Architecture Description Language), and its standardization process is led under the authority of the Society of Automotive Engineers (SAE), aerospace division. The standardization committee includes American and European partners, many of them come from avionics industry. Now, AADL is no more specific for avionics and the meaning of its acronym has changed into "Architecture Analysis and Design Language" to avoid confusion.

In Europe, the ASSERT project is considering the same problems, but is more innovative and ambitious. Using a common architecture description language for the whole development cycle, it aims at improving the engineering process by using formal methods and proofs, and go towards the *proof-based system engineering* (PBSE). ASSERT is an integrated project (IP) for the 6th framework program for research and develoment of the European commission. Many of the large avionics and space European compagnies are involved in this project, like other ones coming from other industrial domains. At the end of this project, the results will have been applied and validated on real industrial systems.

Automotive domain

The car industry has the same considerations as the avionics one. They also have projects based on architecture description language approaches. The French AEE project [aee] has defined some specifications of what is useful for the automotive industry. In particular it has designed a specialised ADL, *AIL-transport* [ES2002], which is more dedicated to this domaine.

Unlike more generic ADLs, AIL-transport details many categories of automotive components. So this ADL is strongly linked with the automotive architecture and cannot be adapted to other purposes. This choice may be ex-

plained by the fact that the development process of a car is shorter than the one of an aircraft.

AIL-transport and the results of this project are internally used by some ot its automotive partners. However, the lack of tools supporting them is a weakness and stresses the need for tools to promote new system engineering practices.

The European project EAST-EEA [east] is in a certain way the follow up of the AEE project. It demonstrates the interest of the automotive industry, at the European scale, for the formal description of architectures and the ADLs.

AADL emergence

Among all the ADLs, AADL is currently emerging and could play a major role in industry in the future. This is illustrated by projects like ASSERT, which has chosen this language to support its system development lifecycle. Several explanations can be advanced:

- AADL is generic enough to cover all the real-time systems, in any domain;
- AADL is currently standardized by an international committee, what favours its adoption by a large community of users and its support by tool vendors;
- AADL is based on MetaH, which has experimented and validated for ten years the underlying concepts;
- a graphical version of AADL, based on UML 2, is currently under specification.

As seen previously, some other domains such as automotive have chosen more dedicated ADLs. It should be possible to map the concepts of these special ADLs on those of AADL. Such an approach could combine the advantages of AADL with specificities of a given domain. This idea is worth studying.

3. Tool support for AADL

Needs

Whatever the quality of a method is, it will not be adopted by users if it is not supported by tools. This is especially the case for development methods in industrial domains, where training and process constraints are important. It is not thinkable to ask the users to write by hand formal descriptions of their systems, and manually translate them into other formats to communicate with their development tools.

This is also the case for AADL. This language provides a strong structure and a well-defined semantics for the description of system architectures. It is

possible to build a good development process with it. But all this will only be done if tools help the users during the development. At the same time, tools will also ensure the correct use of AADL by avoiding mistakes, syntax errors and so on.

Existing basis to build a tool suite

A tool suite supporting AADL does not need to be developed from scratch. Indeed, many tools exist and can be used as a basis to lead such development. Even if some of these tools can not be easily adapted, they demonstrate the feasibility of these concepts.

MetaH. As said before, MetaH is both an architecture description language and a toolset supporting it. The language has been used as a basis to define the AADL standard. Even if differences between them are sometimes important (MetaH has more restrictions for multi-processor systems, scheduling policies, etc.), the common part is large and ensures a strong basis for AADL. When considering the development of tools supporting the new AADL standard, it is natural to imagine the reuse of some MetaH tool technologies.

MetaH provides a graphical user interface called Dome. This interface enables the design of the architecture and its properties, and the generation of the description file. Dome has meta facilities which enable its easy adaptation to support the new AADL concepts. Moreover Dome has not the same restrictions as MetaH for exportation.

MetaH also contains some verification tools for architectures described in MetaH ADL. For example, it is possible to check the schedulability of the real-time tasks. It seems interesting to reuse such a tool with AADL. The schedulability problem is relatively simple with MetaH because the scheduling policy is imposed and all the related theories exist. AADL allows many different scheduling or communication policies. The theories that we need to prove the schedulability of a whole architecture are now more complex, when existing, and the reuse of the MetaH tools is perhaps not so easy.

The MetaH toolset contains code generators. This is due to its general principle, where the user components are agregated and linked by the generated code. In this approach, many constraints can be imposed to the user, since the expected result is the production of some embedded software. The principle of AADL is more a descriptive approach, that is, the capability to describe a possibly already existing system. In this case it is not possible to impose many technical solutions to the user. The MetaH code generator may be reused with AADL but it needs to be adapted to the new context and become more flexible.

Thus, MetaH can be used as a basis for several tools supporting AADL, but adaptations will be required in all the cases because of the differences between

these two languages. The main risk is related to the current export restrictions imposed to MetaH by the US government.

ADeS. ADeS [ades] is a simulator of the behaviour of system architectures described with AADL. We developed it during a study with the European space agency (ESA) to evaluate the first draft versions of the AADL standard and to test the capability of this language to be supported by tools.

ADeS can be used during the development of an architecture to assess its behaviour. At each step of the lifecycle, the AADL description of the architecture is refined. ADeS parses this description and simulates the behaviour of all the components to show the global behaviour of the architecture. The accuracy of the result will depend on the accuracy of the description. On the one hand, during the first phases of the development we just get an overview of the global behaviour. On the other hand, at the end of the development, it is possible to plug user models in the simulator to have precise results. This capability of ADeS to accept user behaviour models is used to bypass current limitations of AADL in the description of architecture behaviours. We expect improvements of this point in the future.

Other tools can be used for similar simulation purposes (e.g., ObjectGeode [objectgeode], Matlab/Simulink [simulink], Scilab [scilab]). What is different with ADeS is the will to share a common architecture description language as a support for the whole information related to the manipulated architecture, including the simulation needs, and not to use a specific language for simulation aspects or particular subsystems. However, we have succeeded in the connection of ObjectGeode and Scilab with ADeS to take advantage of their specific capabilities.

However, ADeS can provide several interesting modules for the development of an AADL tool suite. Firstly, ADeS contains a full AADL parser, able to understand AADL descriptions, to provide a data structure representing the architecture in memory, to detect the syntactic and semantic errors, and to send an explicit error message to the user. This parser needs to be updated to take into account the last evolutions of the standard, but it is already very close to the AADL specification. As foreseen for the ASSERT project, this AADL parser can also be used to experiment changes in AADL to support additional informations.

Secondly, ADeS provides a graphical user interface (see figure 1) to visualize the structure of the system architecture, and not only the structure of the description. In other words, it recursively shows contents of all the component *instances*, whereas many common modelling tools show more the component types (or classes).

Lastly, ADeS has a simulation kernel where simulation elements are created to represent the system architecture elements. The simulation provides a good

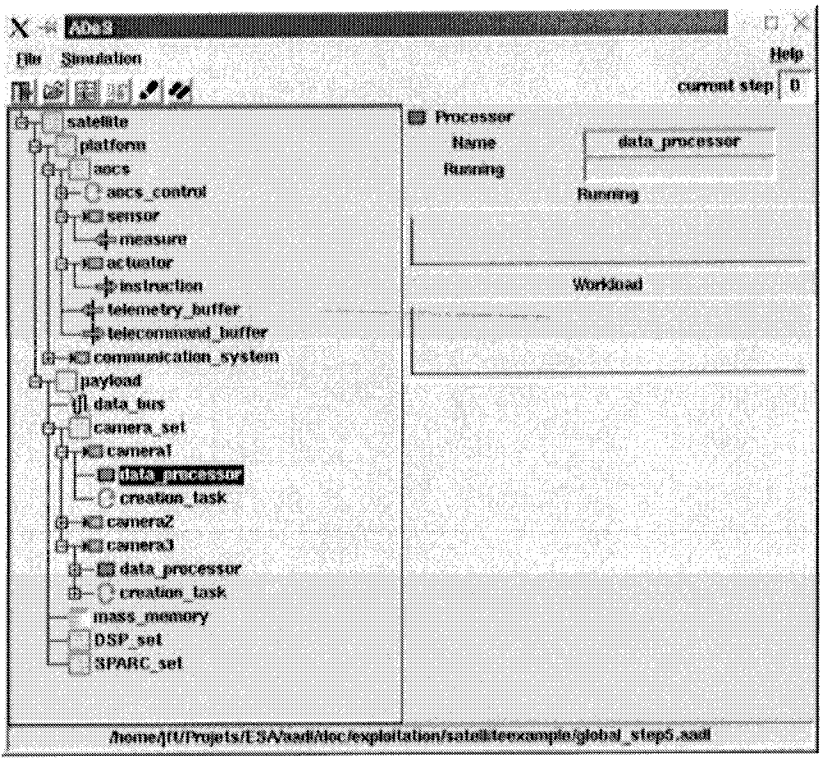

Figure 1. Snapshot of the man-machine interface of ADeS

means to evaluate the behaviour of the system under development. This tool can be extended to support many specific needs of the user.

A possible extension of ADeS is the addition of a capability to modify the manipulated AADL descriptions in order to easily prototype, dimension and assess solutions, and then reexport these changes into the AADL description.

Possible services

Many services can be provided by a tool suite based on AADL, in several categories: modelling tools, verification, automatic generation, etc. These tools will share some common functions, such as the AADL parser. Figure 2 shows a possible organization of an AADL tool suite. Let us consider these possible tools.

AADL manipulation. AADL parsing is the first action for any tool using an AADL description as input. Thus, AADL parser is a typical example of a module which can be shared by all the tools supporting AADL.

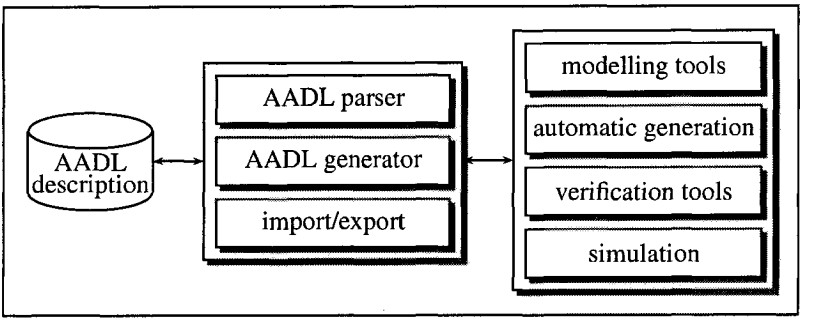

Figure 2. Possible organization of an tool suite supporting AADL

The first objective of an AADL parser is to read the AADL descriptions and to build in memory a data structure which will be manipulated by tools. Such a parser can be applied on textual AADL descriptions or their equivalent XML format, but also on XMI format when using graphical AADL descriptions based on UML. Depending on the context of its use, it can provide several other services:

- Detection of all the syntax and semantics errors. This is useful when the description is written by hand or by a tool which does not correctly manage all the semantics rules of AADL;
- Creation of error messages explaining the detected errors. This is useful when using the textual AADL format in a description written by hand.

Another similar module can be shared: the AADL generator. This module is able to produce an AADL description from a memory representation of the architecture. Of course, these two modules have to work together to share the same memory representation. They constitute a single software library for the tool developer.

Modelling. Many modelling tools exist and are used for system and software development. Some of them are based on UML [uml], others on HOOD [hood], other on different standards. They enable the description of an architecture by combining elements and assembling them. Introducing AADL in the system development process strongly requires such modelling tools. This requirement can be fulfilled either by specific tools developed to support AADL, or by existing ones where some import/export functions are added to read and write AADL descriptions. A graphical version of AADL is currently under definition. This graphical AADL is based on UML 2. This choice will help in the reuse of existing UML modelling tools in this context.

Just using existing modelling concepts is not enough to take completely advantage of AADL. The first AADL descriptions of a system under devel-

opment can happen in the very early stages of a project, for example during the capture of the needs. And the classical modelling tools are not able to take these aspects into account. Some tools exist to support requirement engineering (Doors [doors], Catalyse [catalyze]) and could also be interfaced with AADL. Innovative methods also exist to deal with this problem with formal approaches and proofs of consistency. For example, TRDF [GLL1996] defines how to rigorously lead the first phases of the development to capture all the requirements and to produce complete and consistent specifications. Since these methods are not yet supported by tools, new ones have to be invented. They will enable the production of AADL description during the application of these methods.

Examples of possible tools are given below:

Requirement capture this tool will help in the capture of the need of the client and its formal expression in the highest levels of an AADL description;

System design and validation this tool can help in the description of the specifications of the system and the reuse of already existing building blocks with formal verification of their compatibility with the architecture;

Feasibility and dimensioning this tool will ensure that the already described architecture is feasible, and will complete the description by giving values for the pending properties.

Generation. During a development process, many parts can be automatically generated instead of being fastidiously hand coded. This is the case for the source code itself. This function is now well masterised by tool vendors and is often associated with modelling tools.

Automatic test generation is also possible in this context. Some of these tests are based on the low-level aspects of the develoment. Unitary tests are related to the source code and the detailed design results. A lot of research is led to better cover this activity [inka, danocops, agatha]. By choosing a single formal description language for the whole development cycle, we enable the use of this language to contain what is needed to generate these tests. Moreover, in such a description we also have a lot of information to generate higher level tests for integration phases.

Verification. Since we use a formal means to describe the system architecture under development, many formal verifications can be processed. For example, it is possible to *prove* real-time properties: Is it possible to always execute the tasks and meet their deadlines? Is it possible to communicate all the data through the connections? Such verifications will take place after a

detailed design phase because they need precise information about the organization of the system and its parameters.

Another type of verification can be performed during the first phases of the development (capture of the needs, specifications, design). Any of these phases will produce and refine the AADL description of the system architecture. Provided that the development method keeps all the needed information in the description, it is possible to check by tools whether the architecture resulting of each phase is consistent. It is also possible to check whether all the requirements resulting from a given phase are taken into account in the next one.

Simulation. Verifications can also be performed by simulation of the execution of the system architecture under development. If we have all the interesting information in the description, it is possible to simulate the behaviour of each element and get a good evaluation of the global system. As presented previously, ADeS is an example of such a tool and illustrates the capability of AADL to provided the needed information to perform such a simulation.

The simulation can also provide an evaluation of the behaviour of the architecture during each step of its development. This is useful for a rapid prototyping of the system during its specifications or at the beginning of its design. As the design is refined, the simulation can be used to evaluate its consequences on the behaviour of the system.

4. Conclusion

We have seen that ADLs are more and more considered, and particularly in avionics and automotive domains. Among these ADLs, AADL seems to be an interesting one, and the development of a tool suite supporting this language is now necessary to promote it and enable its easy use by industry. Bases exist to ensure the feasibility of these tools and to build them by reusing some components.

Tools supporting AADL can provide services in several categories: A set of parsers and AADL generators can help in the manipulation of the language, modelling tools can help in the creation of architecture descriptions and provide support for innovative formal design methods, automatic code and test generators can take advantage of the information contained in the description, verification tools can check the consistency and completeness of the designed architecture, simulation tools can give an assessment of the behaviour of the system architecture during its development.

Since the use of AADL in system engineering may be very large, a complete tool set must be composed by many more or less independant tools, developed or adapted by their own specialists. The difference with the current situation

will be the interoperability capabilities, thanks to the international standardization of AADL which enables these tools to share the same language.

The development of innovative tools and methods in relation with research projects, like ASSERT, will provide improvements for AADL. The standarization committee has already planned a new version of the standard to take into account these inputs.

References

[aadl] Axlog ingénierie, "AADL (Avionics Architecture Description Language)", 2003, http://www.axlog.fr/R_d/aadl/aadl_en.html.

[metah] Honeywell, "MetaH Evaluation and Support Site", 1998, http://www.htc.honeywell.com/metah/.

[aee] AEE, 2002, http://aee.inria.fr/.

[ES2002] Jean-Pierre ELLOY, Françoise SIMONOT-LION, "An architecture description language for in-vehicle embedded system development", 2002.

[east] "EAST-EEA – Embedded Electronic Architecture", http://www.east-eea.net/.

[ades] Axlog ingénierie, "ADeS", 2003, http://www.axlog.fr/R_d/aadl/ades_en.html.

[objectgeode] Telelogic, "Telelogic ObjectGeode", http://www.telelogic.com/products/additional/objectgeode/index.cfm.

[simulink] Mathworks, "Simulink", http://www.mathworks.com/products/simulink/.

[scilab] Inria, "Scilab", http://scilabsoft.inria.fr/.

[uml] "Unified Modeling Language", http://www.uml.org/.

[hood] HOOD User's Group and Jean-Pierre ROSEN, "HOOD, An Industrial Approach for Software Design", 1997.

[doors] Telelogic, "Telelogic DOORS/ERS", http://www.telelogic.com/products/doorsers/index.cfm.

[catalyze] SteelTrace, "Catalyze suite", http://www.steeltrace.com/products_catalyze_suite.htm.

[GLL1996] Gérard LE LANN, "Proof-Based System Engineering and Embedded Systems", invited paper, European School on Embedded Systems (Veldhoven, NL, Nov1996), in Lecture Notes in Computer Science nř1494, Stringer-Verlag Pub., Oct 1998, pp. 208-248. http://www-rocq.inria.fr/reflecs/publications_fr.html.

[inka] "Projet InKa", http://www.axlog.fr/R_d/inka/inka_cadre.html.

[danocops] "DANOCOPS : Détection Automatique de NOn-COnformités d'un Programme vis-à-vis de ses Spécifications", http://www.telecom.gouv.fr/rntl/projet/Posters-PDF/RNTL-Poster-Danocops.pdf.

[agatha] "Atelier de génération de tests de spécifications industrielles AGATHA", http://www-drt.cea.fr/fr/prog/list/systemes_embarques/list_outils_atelier.htm.