A. A. Koltashev
JSC "Academician "M. F. Reshetnev "Information Satellite Systems", Russia, Zheleznogorsk

A PRACTICAL APPROACH TO SOFTWARE PORTABILITY

This paper describes an approach to porting onboard software for communication and navigation satellites to new platforms that use various onboard computers and devices. The approach relies on the target(onboard) and the tool software stratification and strong typing and the Modula-2 programming language especial features.

Keywords: software engineering, satellites onboard software, cross-programming system.

The approach is being resolved by the JSC "M. F. Reshetnev "Information satellite systems" in collaboration with the Ershov Institute of Informatics Systems and the Excelsior, Ltd. (formerly XDS), via an approach based on architectural stratification and interface standardization both for the onboard software and the development environment and the Modula-2 programming language especial features [1], such as strong typing and separate compilation [2–7].

First of all, a stratification of the OSW was performed which allowed us to define and implement an OSW Abstraction layer that provides a standard platform-independent API (types and procedures) to application software (i. e. the software of satellite subsystems that solves functional problems and amounts to up to 80 % of the entire OSW). Portability of the application software to new hardware is ensured by the software development system whereas the use of a new operating system and drivers, by reprogramming the implementation modules for the OSW Abstraction layer.

Next, a stratification of the cross-programming system (CPS) was made which allowed us to define platform-independent and programming-language-oriented user interfaces: both for programming and testing/debugging needs, and the CPS Abstraction layer that provides a standard architecture-independent API (types and procedures) that isolates architecture–dependent parts of the onboard computers: CPS components implement code generation and instruction set simulator.

A successful implementation and an efficient use of both the OSW and CPS Abstraction layers proved to be possible to a great extent due to the strong typing and separate compilation properties of Modula-2.

Moreover, the use of a highly structured language provided an additional benefit, namely, a possibility to measure static and dynamic software component characteristics and to calculate criterions of software testing completeness for quality assurance.

Onboard software stratification. A canonical three-layer structure of the OSW was determined. The third layer – the application software – becomes platform-independent because its implementation only uses interfaces exported by the lower, second layer called the Abstraction layer. The first layer is the employed operating system, i. e. a real-time kernel and a set of drivers.

The canonical structure of the OSW Abstraction layer for the given problem area was determined by the analysis of functionality of several generations of satellites.

Three components are specified in the Abstraction layer:

– standard interfaces for a canonical set of abstract real-time kernel calls;

– standard interfaces for a canonical set of abstract onboard devices;

– standard types for a canonical abstract data set used to control subsystems of the satellite in various regimes.

The main features of Modula-2 – the strong typing and separate compilation – gave a possibility to implement the Abstraction layer as a set of Modula-2 libraries providing a complete and platform-independent API. It proved possible to define the data types independently of addressability, endian-ness, word size, etc., of various onboard computers.

The strong typing, e.g. enumeration types, provided a required level of abstraction for data types which is both maximally close to the problem field and also platform-independent to a maximal degree.

The separate compilation feature allowed us to standardize and freeze the API using definition modules, and made it possible to reuse all the functionally equivalent application software without any changes in the source code, which greatly simplified configuration management.

A Modula-2 cross-programming system adaptable to different target onboard computers. We developed a cross-programming system (CPS) based on Modula-2, which implements the code generation for some base computer implemented via an instruction set emulator. In this CPS, there were implemented platform-independent programming-language-oriented user interfaces as well as the CPS Abstraction layer that provides a standard architecture-independent interface with parts that depend on the architecture of the target onboard computer: the CPS components that implement code generation and the instruction set simulator.

Adaptation CPS for a new embedded computing system is performed by creating code generation and the instruction set simulator components only. The CPS ensures that all debugging information is provided at the level and in terms of Modula-2.

The assembly of the project into executable code is done via a standard CPS shell with the use of the necessary components of the commercial software development system. Another important fact is that application programmers can notice whether a switch to a different target computer (another command set interpreter) has occurred only through changes in the program operating characteristics as measured by the CPS, because all the user interfaces stay exactly the same.

The CPS also includes testing and debugging facilities that use platform-independent testing languages. The platform-independent dialogue and batch testing languages allow programming, executing and documenting test

procedures in the platform-independent terms of the programming language. The batch testing language allows one to reuse the test procedures with another platform.

The CPS runs efficiently under MS Windows 2 000/XP on Intel Pentium chips.

Such an approach was influenced by the XDS programming environment [5], and the XDS system was used as a programming environment for the CPS development and maintenance.

So, the architectural stratification of the OSW and CPS and the standardization of the two Abstraction layers ensures an easy adapation of the CPS to a new target computing platform.

An additional benefit of using a highly structured language. Our CPS allows one to obtain a complete set of measurements for the program unit being developed (including the size of stacks, the execution time, etc.) in order to evaluate some measures of source code quality and to calculate the C1 (all branches) and C (all decisions) criterions of unit testing completeness.

This is only possible with a highly structured language, so this quality allows one to implement in the CPS a complete check of the program execution process during the testing procedure and to make the testing procedure (especially under batch testing) independent of local changes in the source code of the program being tested.

The described technology of the OSW development results not only in a high level of reuse and portability of the OSW, but also a possibility to reuse all testing procedures for regressive tests of the OSW.

This is true not only for the OSW development, but also for the CPS. Moreover, it is possible to start the OSW development not waiting for adaptation of the CPS to a new target onboard computer by using the CPS available for the base onboard computer.

The measurement tools of the CPS that are based on the properties of the programming language allow one to achieve the high level of quality of the OSW components required for the spaceflight applications.

The standard and stable user interfaces of both the CPS and the Abstraction layer greatly simplify the maintenance of OSW by facilitating experience accumulation and transfer.

The technology successfully worked under development onboard software for Exspress-AM series communication satellites and for Glonass navigation system's satellites.

Bibliography

1. Вирт, Н. Программирование на языке Модула-2 : пер. с. англ. М. : Мир, 1987.

2. Koltashev, A. A. Practical Approach To Software Portability Based on Strong Typing and Architectural Stratification : Joint Modular Languages Conf., JMLC 2003. Klagenfurt, Austria, August 25–27, 2003. Proc. Lecture Notes in Computer Science (LNCS 2789) / A. Koltashev. Berlin : Heidelberg : N. Y. : Springer-Verlag, 2003. P. 98–101.

3. Колташев, А. А. Технология переноса бортового программного обеспечения / А. А. Колташев // Открытые системы. 2004. № 4. С. 3.

4. Колташев, А. А. Технологические аспекты создания бортового программного обеспечения спутников связи / А. Н. Антамошкин, А. А. Колташев // Вестн. Сиб. гос. аэрокосмич. ун-та : сб. науч. тр. Красноярск, 2005. № 6. С. 93–95.

5. Native XDS-x86 (User's guide) // The XDS product family. XDS Ltd. 1997.

6. Средства измерения бортового программного обеспечения / А. В. Еремин, О. С. Иноземцева, А. А. Колташев и др. // Вестник СибГАУ. Вып. № 1 (18). Красноярск, 2008. С. 52–56.

7. Колташев, А. А. Современная технология разработки и сопровождения бортового программного обеспечения спутников связи и навигации / А. А. Колташев, С. Г. Кочура, В. В. Хартов // Космические вехи : сб. науч. тр. // ОАО «Информ. спутниковые системы» им. акад. М. Ф. Решетнева». Красноярск, 2009. С. 237–251.