

/k/ Embedded Java Solutions

Putting Java where it wants to be

Embedded Systems and Java

Embedded systems are changing. Not so long ago, “embedded” meant “isolated” — an embedded system would communicate with some specialised hardware, maybe with a desktop or mini-computer (via its own unique protocol), or just maybe with a human operator via a control panel. Sometimes the embedded system would communicate with others of its own kind; often custom interfaces needed to be designed in order to get data from one kind of embedded system into another. Each embedded system had its own unique programming environment, and code developed for one system was useless to any other. Adding new functionality was something the manufacturer did in the next release of the firmware.

Now, embedded systems are increasingly interconnected — not just with other similar devices, but with quite different devices and with computer networks. Ad-hoc interfaces are replaced by standard protocols such as CAN or ProfiBus, or by ethernet and TCP/IP. At the same time, the hardware is becoming more powerful, leading to the use of higher-level programming languages, and also becomes easier to re-program — PROM becomes UVROM becomes EEPROM. It therefore becomes natural to look at the possibility of using the network to download code (functionality) to an already installed system, and to ask whether this code could not be re-used or even shared between different devices.

The advantages

Looked at in this way, the prospect of being able to use Java for downloadable code looks attractive:

- Modern object-oriented language, but already in use for almost ten years. Popular for teaching and for self-study, so basic competence is easy to find.
- Designed-in portability; WORA may be more of a goal than an achievement, but it's still a worthwhile goal.
- Bytecode and JAR format are quite compact - generally many times smaller than equivalent native code.
- Code distribution and security mechanisms which have been tested in “battlefield” conditions on the WWW.
- Rich set of APIs and a growing portfolio of 3rd-

party libraries.

The challenges

Still, many companies hesitate to incorporate Java into their embedded systems, citing concerns about:

- Resource usage.
Yes, Java can require more resources than C, just as C can require more resources than assembler. But Java can also bring huge improvements in programmer productivity and in reliability, just as C does in comparison to assembler. And there are products from specialised vendors such as Aicas which can rival native code in size and speed.
- Licensing costs.
Many embedded Java vendors charge per-unit royalties which are hard to accommodate in an embedded system's budget. But there are also open-source products which can form the basis of an embedded Java solution, such as the gcj Java-to-native compiler or the Wonka embedded VM. /k/ can help you optimise these for your system.
- Incompatible and unstable APIs.
Certainly Java took some twists and turns in the early days, and some embedded Java products turned out to be an expensive dead end. But with the emergence of J2ME the relationship between embedded and desktop Java has become much clearer, and a wise choice of APIs can avoid the risk of built-in obsolescence.
- Lack of in-house experience.
This will always be a problem when adopting a new technology. The solution is to work with an experienced partner such as /k/ during the initial design period, while your staff have a chance to familiarise themselves with the technology based on hands-on experience with the emerging product.

The solution

In short, embedded Java is a technology which is ripe for incorporation into your company's products, but you will want to take the first steps with an experienced partner. /k/ Embedded Java Solutions is an independent consultancy specialising in embedded Java, with the know-how to advise and assist you. Contact us on k-info@kiffer.be or visit our web site, <http://www.kiffer.be/k/>.