ProJor business use-case

Company Background

ModuWare Systems Ltd. maintains a **plugin-based** field-service **platform** (150+ modules) for utility companies. Each module exposes its metadata through a lightweight JSON descriptor consumed by the runtime.

Their Vision

A new product generation mandates that every module exposes its capabilities as a strongly-typed Java class that implements an annotated interface. Migrating 150+ modules manually would freeze feature work for weeks and endanger release dates. ModuWare wants this structural shift - and any future one - to be performed safely and in bulk.

They also envisioned developing a self-serving feature development agent for their customers, so they are able to create the modules they need themselves.

Current Environment

The ModuWare application lives inside a mono-repo, which contains the application core and all of the application's modules. There is a lot of code and logic duplication, which was technically necessary for the module-based approach to be viable: JSON descriptors and module build configs are routinely copy-pasted to create new modules, while realistically there is only a few lines of code difference between the modules.

Using LLMs directly to generate code in the new self-serving developer agent turned out to be a bad idea due to hallucinations, that seem to stick around nomatter the contents of the system prompt.

Business Requirements

- Migrate all existing modules to the new Java-class descriptor
- Avoid long freeze periods during future framework upgrades
- Develop a self-serving module developer agent for their customers

Technical Requirements

- Model module structure and descriptors explicitly as code
- Generate:
    - Java descriptor classes implementing the annotated interface
    - Updated module wiring and dependency injection configurations
    - OpenAPI contracts of the modules
    - Module registration and integration tests
    - Consistent build configurations across all modules
    - Up-to-date developer documentation (e.g., module catalog, architectural diagrams)
- Allow changes in descriptor structure to be propagated automatically to all modules
- Maintain clean separation between generated scaffolds and hand-written business logic

Application of ProJor

ModuWare uses ProJor to formalize the internal structure of **modules**, including their descriptors, wiring, and build system definitions.

The descriptor definitions, build configuration files, integration tests, and developer documentation are all generated from the ProJor model, which eliminates duplication and guarantees consistency across the entire mono-repo.

When the structure of a module descriptor changes, the model is updated once, and ProJor automatically regenerates the affected scaffolding across all modules, without touching custom business logic. This drastically reduces the cost and risk of future framework upgrades.

Furthermore, the self-serving agent can now generate the YAML input of ProJor, resulting in 100% complete and working project scaffolds, opposed to hallucinated or incomplete outputs.

Summary

By adopting ProJor, ModuWare Systems transforms its plugin ecosystem into a fully model-driven architecture. They eliminate the risk and manual labor traditionally associated with large-scale structural upgrades, enabling mass migrations and framework evolution with minimal downtime.

New module descriptors, wiring, build configurations, tests, and documentation are generated automatically from the central model, keeping the system consistent and traceable at all times.

The introduction of model-driven inputs for the self-serving module developer agent ensures that customers can create new modules reliably, without the instability and unpredictability of LLM-based freeform code generation.

As a result, ModuWare not only safeguards the long-term maintainability of their platform but also unlocks new revenue streams by empowering customers to extend the system themselves — securely, quickly, and without compromising architectural standards.