MC-3020 Static Instance Population

Overview

Software systems prefer to awaken from reset to better than a vacuous universe. Processing and control systems place expectations on data that are greater than empty abstract specifications. Beginning with some few concrete samples and examples of interrelated class instances provides a significant head start to a program. Some things can be presumed to exist at system start-up, namely, collections of instances with preset attribute values which are not created with explicit, dynamically executed actions. These collections simply are there at power up.

Analysis

The set of preexisting instances with attributes and relations must be specified by the system analyst. A means must be provided to clearly and easily establish as input to translation the set of preexisting instances. Additional machinery must also be supplied within the compiler to generate target code integrating static instance population.

Requirements

1. Means of specifying preexisting class instances.
2. Means of specifying the attributes of specified preexisting class instances.
3. Means of specifying preexisting relationships between specified preexisting instances.
4. Means of statically establishing preexisting instances in a target system with little or no dynamic run-time code execution.
5. Means of specifying a preexisting event(s).
6. Means of generating code for this preexisting event(s).
7. Means of exchanging instance information with other BridgePoint tools and DesignPoint architectures.
8. Means of representing and exchanging state save information.

Instance Specifier

A tool must exist to allow a user to specify preexisting instances. Ideally, this tool would be intuitive and easy to use. Customers could clearly and concisely specify the instance data, relationships and attribute values.

Possibilities for this tool include:

- Vi, E3, Emacs, Notepad
- Excel, Lotus, Oreo
- Visual Basic GUI or wizard
- PHP web front end
- Perl/Tk GUI
- Qt-based GUI

This tool at minimum must allow instances with attributes to be listed.

Relationships could be shown graphically or by referential attributes in a relational manner. In either case, it will be assumed that the data provided as input to the model compiler will use the latter.

Data Interchange

A standard format for instance data imported by model compilation is proposed. This format would not preclude other formats but would serve as a common data protocol shared by compliant architectures. Any compliant Instance Specifier tool would need to export data in the standard format.

XML is becoming a defacto standard for data interchange in documents, in data exchanges on the Internet, in embedded systems and elsewhere. It is proposed that XML serve as the preferred data exchange format for instance information.

A minimalistic XML for instance data would include two elements, Instance and AttributeValue. Properties for these elements would roughly or identically parallel the attributes in a metamodel enhancement to support specification of instance data.

Model Compiler Process

Two tables may be added to the architectural OOA metamodel. A table called Instance and a table called Attribute Value are added to the Object subsystem. One Object is related conditionally to many Instances. One Attribute is related conditionally to many Attribute Values. One Instance is related to many Attribute Values.

An instance of Instance exists for each instance in a system. For our purposes, there would be one instance of Instance for each preexisting instance in the system. Each would be properly related to the correct instance of Object. There would be an instance of Attribute Value for each attribute of each instance of Instance. These Attribute Values would be linked to corresponding Attribute instances.

Some time before (second pass) translation, these preexisting instance instances and attribute value instances must be populated. During (second pass) translation, all of the information necessary to populate the static initializer templates would be available. Archetypes must be provided by each architecture to accomplish the code generation.

Archetypes

The archetypes for the static initializers of preexisting instance data will vary between model compilers. Most challenging in this static population is that of linking instances. Architectures that optimize relationships with instance pointer information must predetermine the addresses of correct related instances. Even more challenging is correctly initializing the addresses of container instances when dealing with collections of instances as required when multiplicity is greater than one.

The archetypes and archetype helper functions used to create and delete relationships serve as a foundation for the initializer archetypes.