Managing type information in an evolving environment

V. Mencl, P. Hnětynka

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. When developing component–based software, certain meta–information describing the component types has to be developed together with the components. This paper proposes a *Type information repository (TIR)*, designed for storing the meta–information as a database with an hierarchical namespace and a distribution–aware versioning model.

The TIR will provide support for concurrent development by multiple development teams; the version model developed for the TIR will ensure consistency among the TIR nodes.

Introduction

When developing component–based software, certain meta–information describing the component types has to be developed together with the components. This metainformation is traditionally entered in the form of textual descriptions, however, this approach by itself suffers from certain drawbacks.

Applications are expected to be assembled from components developed by multiple vendors; this development may be concurrent. During such development, interfaces of the components may evolve, regardless of prior negotiation of the final form of the interfaces.

In order to allow for consistent handling of concurrent development of software components (eventually with evolution of component interfaces), a facility for accessing the metainformation in a shared way is necessary.

Such facility should provide mechanisms for concurrent development of the metainformation at a single development site, and mechanisms for coordinating development among remote development sites (although set of features may be reduced for remote interaction).

To allow for proper coordination of the development, an sufficiently mature version model should be employed. Multiple versions of an interface need to be developed concurrently, this results into a need of *branches* in the version model. We have formulated criteria for the version model, and developed a version model fulfilling these criteria.

To satisfy the needs described above, we have developed a *Type Information Repository (TIR)*. This repository is capable of storing generic descriptions expressed in a description language, with special focus put on description languages employed in the SOFA project. Each particular language requires an extension module for the TIR, currently, the CDL language [MENCL98] is supported.

SOFAnet – the environment of the TIR

SOFA is a project of the *Distributed systems research group* at the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University. The goals of the SOFA project are to provide a component model suitable for easy and reliable composition of reconfigurable applications from components; SOFA itself stands for **SOF**tware **A**ppliances.

SOFA uses a hierarchical component model, i.e. components can transparently encapsulate other components.

SOFA assumes existence of a global network (*SOFAnet*), consisting of autonomous nodes (*SOFAnodes*). SOFAnodes are capable of holding software components, transferring these components among themselves, and where appropriate, deploying the components in a runtime environment. Each SOFAnode has a *Template repository (TR)*, the storage of SOFA components, and may contain a *Made*, *In*, *Out*, *Run* part, used for creating, resp. receiving, resp.

transmitting, resp. running the components. Figure 1 shows the scheme of several SOFAnodes. For a detailed description of the SOFA architecture, please refer to [PBJ98].

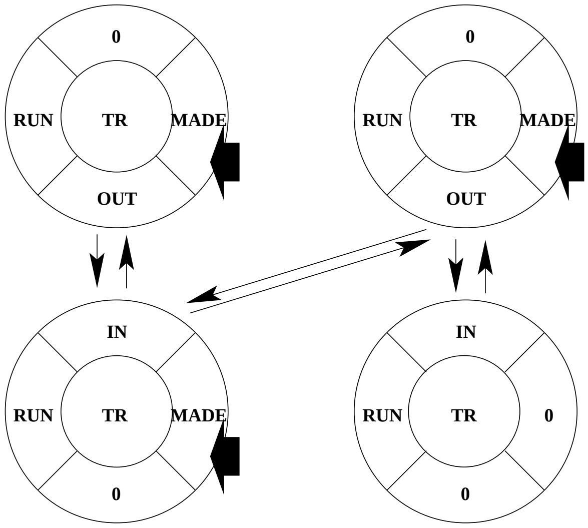

Figure 1. The scheme of the SOFAnet

The SOFA languages

In [MENCL98], the *Component Definition Language (CDL)* was proposed. Several other description languages used for specifying various aspects of software components are used in SOFA, these include the Update Description Language (UDL), the Component Implementation Description Language (CIDL) and the language for deployment descriptors. As the SOFA project is under progress, further languages may appear; therefore, the repository should be extensible.

The CDL language is similar to the OMG IDL language, CDL declares interfaces of services, components (black–box view of a component), and component architectures (gray–box view of a component). Definitions declared in the CDL language are used in both the component providing a service and the component requiring a service, as the development of these components may be performed at different locations, a way for sharing the information used in both the components is essential.

The Made part of the SOFAnode

The Made part of the SOFAnode is responsible for creation of new components. The Made part accesses the TR of the SOFAnode, and submits finalized components to the TR, making them available either for deployment in the Run part, or for transfer to another SOFAnode.

When a component is packaged, the implementation of the component must be accompanied by certain meta–information – the description of the component's interface, as well as directions for the deployment of the component. These description can be either created anew with the component, or they may be read from the TR, if these were already developed before, and arrived to the SOFAnode with some other component.

The Type Information Repository

To provide for the requirements stated in this paper, the Type Information Repository has been developed. The TIR is to be accessible by all entities of the Made part, regardless of what component-development tools are used in the Made part. The scheme of use of the TIR is shown in figure 2.

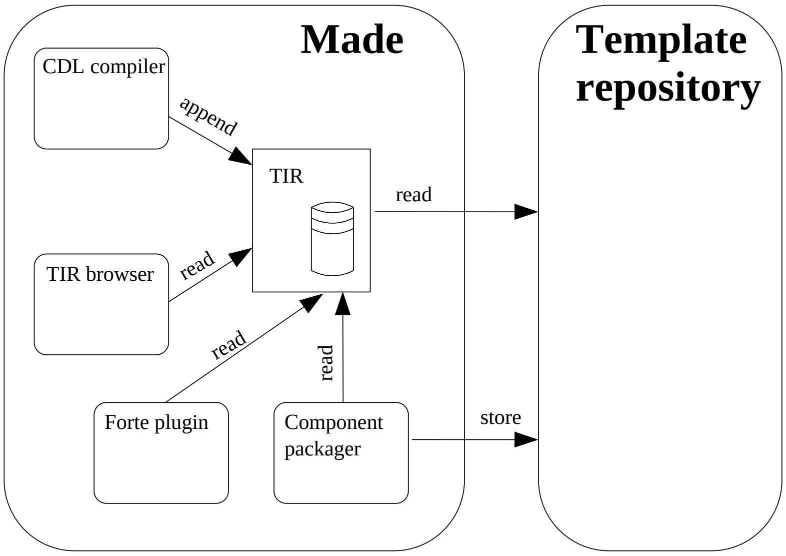

Figure 2. The scheme of use of the TIR

Version model used

It is not within this scope of this paper to describe the version model in detail, only the criteria stated for the selection of a version model will be presented here.

In the SOFA environment, each object declared is uniquely identified by its name (located in a hierarchical namespace) and a version identifier. For the purposes of the declaration languages, the version identifier is an arbitrary string, the version model has to define the semantics of the version identifiers.

As the distributed nature of the SOFAnet dictates, declarations of interfaces can be concurrently developed on multiple SOFAnodes with no direct communication; the only communication taking part among SOFAnodes is exchange of packaged components.

Also, on a single node, multiple *branches* of an SOFAnode may be concurrently developed.

From the needs of the SOFA environment, the following operations must be supported by a version model:

equals: is is essential that two version identifiers can be compared whether they equal

create–next: for the purposes of evolution of an object, support for creating a direct descendant of the object (in the version hierarchy) must be provided. It must be ensured, that each object has at most one descendant.

create–branch: to allow more than one new versions to be derived from a single version, support for creating a branch must be provided. The branch must be uniquely identified.

create–initial: to avoid the chicken–and–egg problem, support for creating an initial version of an object must be provided.

When searching for a version model fulfilling these criteria, the following rules for the version model were discovered:

1. Each version identifier consists of three elements:

(a) identification of the node which created the version identifier *(It is assumed, that each node has a unique identification.)*

(b) identification of the parent branch (may be null, in case of the *main* branch). The identification of a branch consists of the version identifier of the revision from which this branch originated, identification of the node and the sequential number of the branch (among all branches originating from the same revision at this node).

(c) sequential number of the version in the branch

2. For every version identifier, the create–next operation may be used at most once.

3. The create–next operation may only be apply to version identifiers created locally (at the same node). To modify remotely created revisions, a local branch must be created. (As a consequence of this, branches do not span across multiple SOFAnodes).

An implementation of a version model developed according to this criteria is a part of the TIR. The mapping of the version identifiers to data stored as strings is a part of the TIR interface.

The TIR architecture

The TIR has been designed as a hierarchical storage, capable of containing meta–information objects of multiple types, keeping multiple versions of the objects, and thus maintaining the version database. The types of meta–information objects currently supported are those which are known by the CDL language, however, the repository has been designed in an extensible way, and support for other languages can be easily added.

As the SOFA infrastructure is implemented in the Java programming language, Java was used also for the TIR. As the transport mechanism for remote access, the Java *Remote Method Invocation (RMI)* framework was used. The TIR repository server makes itself available to RMI requests, clients (e.g., the CDL compiler) connect to the repository, and access objects running in the server's address space. The repository can be browsed; to modify the repository, a transaction has to be begun, within the context of the transaction, modifications can be made, which are stored only and only if the transaction is committed.

The browsing interface of the TIR has been designed similar to the CORBA *Interface repository*; to support specific needs of SOFA, information may be accessed either remotely, or a serialized form of the meta–information may be transferred to the client (e.g., to be either stored, or processed off–line).

Related work

There are existing frameworks, which do some of the tasks done by the TIR, however, none of them satisfied our needs. As a part of the OMG CORBA specification, an Interface repository is defined [OMG-IR]. This Interface repository does provide means for browsing meta–information about services provided (i.e. interfaces of the services), however no support for modification of the information is standardized. Also, use of multiple versions is not available yet – although *RepositoryIds* of interfaces contain a version number, use of this feature is not standardized yet, and in all implementations, only a single version (denoted by "1.0") is used.

In the Microsoft's COM/DCOM environment, the metainformation is stored in the windows registry. The concept of registry is similar to our TIR, however, no support for versioning is provided. Once an interface declaration is compiled and stored in the registry, the interface may not be modified, and only new interfaces (i.e. having a different name) may be compiled. Also, the way the unique identifiers are created is misconceptional – the identifiers are of fixed size, and are created from the host identification, current time, and random. Although such chance is very unlikely, same identifiers might be created – especially, after the fixed space is exhausted – either by time or by number of objects stored.

For these reasons, we consider an unbounded version identifier a more suitable solution.

Current status & future work

The TIR has been developed as the main task of a master thesis, and is planned to be submitted for defense in two months. The interfaces of the TIR has been specified, and the CDL language elements have been implemented in the TIR.

The TIR has been implemented, and as a proof–of–the–concept, a CDL compiler, storing the compiled results in the TIR, is nearly finished.

In the future work, the TIR interfaces and implementations will be extended with support for other SOFA description languages (the UDL, CIDL, DD, possibly a language for declaring *connectors*).

Also, work will be done on other clients accessing the repository. These will include a code generator (creating skeletons for component implementations), and a visual browser, created as a plug-in to the Forte development environment.

Conclusion

In this paper, basic results of our effort to create a tool for managing type information in an evolving environment, the Type Information Repository, have been presented. While creating the TIR, a version model was developed (brief description of the version model has been presented here). For more information, please see the website of our research group `http://nenya.ms.mff.cuni.cz/thegroup/`.

References

[OMG-IR] OMG: *OMG Interface Repository*, `ftp://www.omg.org/pub/docs/formal/99-07-14.pdf`

[PBJ97] F. Plasil, D. Balek, R. Janecek: *DCUP: Dynamic Component Updating in Java/CORBA Environment*, Tech. Report No. 97/10, Dept. of SW Engineering, Charles University, Prague.

[PBJ98] F. Plasil, D. Balek, R. Janecek: *SOFA/DCUP:Architecture for Component Trading and Dynamic Updating*, ICCDS'98, May 4-6, 1998, Annapolis, Maryland, USA

[GERGIC99] J. Gergic: *A Versioning Model for SOFA/DCUP Architecture*, Master thesis 1999, Advisor: Frantisek Plasil

[PETROVA99] Z. Petrova: *Update description language*, Master thesis 1999, Advisor: Vladimir Mencl

[MENCL98] V. Mencl: *Component Definition Language*, Master thesis 1998, Advisor: Nguyen Duy Hoa

[CVS] CVS: The Concurrent Versions System, `http://www.loria.fr/~molli/cvs-index.html`