Supporting Architectural Composition at Runtime with π-ADL.NET

Zawar Qayyum, Flavio Oquendo
Université Européenne de Bretagne
Université de Bretagne Sud – VALORIA – BP 573 – 56017 Vannes Cedex – France
zawar.qayyum@univ-ubs.fr, flavio.oquendo@univ-ubs.fr

Extended Abstract

1. BACKGROUND

Architecture Description Languages (ADLs) have been around for sometime, with the objective of providing a style-specific or general semantic framework for describing software architectures. Influenced by building architectures, ADLs initially tended to focus on the structural aspects of software architectures while largely ignoring behavioural ones. While it is true that behavioural aspects of software systems are more relevant at the lower levels of detail than the high-level view, the present day context of distributed and concurrent computing and its various manifestations in service-oriented architectures make behaviour an integral component in high-level architecture design, due to a) the fluid nature of system elements and b) the influence of runtime comportment on their interrelationships.

π-ADL attempts to capture the dynamic needs of such systems by presenting a structure and behaviour description language based on the process formalism of typed π-calculus. System components can be dynamically instantiated and can modify their interconnection at any stage of execution through connection unification and mobility.

Concretizing π-ADL, π-ADL.NET is a compiler and virtual machine of π-ADL for the .NET platform, designed with the objective of rapidly prototyping and simulating software architectures using π-ADL. With the help of π-ADL.NET, we demonstrate the robust support for architectural dynamicity in π-ADL, and the flexibility it grants in system conception.

2. OVERVIEW OF π-ADL.NET

π-ADL.NET is a .NET implementation for π-ADL, developed with the objective of experimenting with π-ADL on a mainstream software technology platform. By generating an implementation level executable, π-ADL.NET extends the scope of π-ADL from an abstract design-level notation to an architecture-centric approach to system implementation. With π-ADL.NET we attempt to:

- preserve the architectural integrity of the system at the implementation level;
- support analysis of the concrete architecture;
- support evolution of the implementation while enforcing its architectural integrity;
- and directly use the implementation mechanisms of the hosting platform.

Execution in π-ADL.NET takes place in *behaviours*. Reusable behaviour templates exist in the form of abstractions, which can be pseudo-applied in line with the π-calculus formalism.

3. SPECIFYING COMPOSITION AT RUNTIME

π-ADL.NET supports straightforward constructs for the composition of executable elements at different levels of details. Within behaviours, code blocks can be composed in parallel using the compose construct:

```
composeBlock := "compose {" block [" and " block]+ "}"
```

Message passing between these concurrent threads of execution is supported by shared connections declared within their containing behaviours. In order to compose different units of execution, pseudo-applications in conjunction with connection renaming can be applied:

```
pseudoApplication :=  "via" Abstraction "send"
          (typeName | value) [renamingClause] ";"
renamingClause :=  "where" "{" [connectionRename]*
          lastConnectionRename "}"
connectionRename := lastConnectionRename  ","
lastConnectionRename := identifier "renames" identifier
```

Renamed connections enable different behaviours to pass data between each other. The ability to compose architectural elements at inter- and intra-behavioural level results in a powerful semantic base for modelling architectural composition at runtime. The two techniques can also be seamlessly combined, as show in the following code snippet:

```
compose {
          via a1 send Void where {x renames c1};
  and     via a2 send Void where {x renames c2};
  and     via a3 send Void where {x renames c3};
}
```

The compose block enables the initiation in parallel of the pseudo-application of the three abstractions *a1*, *a2* and *a3*. The connection *x* belongs to the behaviour containing the compose block, and is instrumental in linking together the connections *c1*, *c2* and *c3* from the abstractions *a1*, *a2* and *a3* respectively.

4. FOR FURTHER INFORMATION

*www-valoria.univ-ubs.fr/**ARCHLOG**/*