Framework for Dynamic and Automatic Connectivity in Hierarchical Component Environments

Gabor Paller

gabor.paller@nokia.com

2005.11.29 , Fractal workshop, Grenoble, France

NOKIA
Connecting People

Fractal in Nokia Research Center

- From product point of view, Nokia is committed to OSGi
- OSGi component model has very strong relationships with Fractal – not by chance.
- OSGi component model is not hierarchical
- Fractal is used for research purposes in NRC

Motivation

- Context-aware middleware feeds context events into the system
- Handling these events with monolithic middleware yields very complex software. More if-s with each new context event handled. This yields tangled code and large footprint
- Reflective middleware is the solution – application and middleware architecture changes
- Dominant approach: dynamic component-based system. Dynamic: component lifecycle and binding changes during the execution of the program
- How do context events translate to architecture changes?
- Dominant approach: event-condition-action pattern and dedicated reconfiguration managers (components which incorporate reconfiguration logic)
- In hierarchical setting: hierarchical reconfiguration managers, one for each domain.

Dynacomp approach

- No application-specific reconfiguration manager. There is one general reconfiguration manager per domain (domain=composite component) which connects components according to meta-information exposed by the components
- Meta-information is also used to move the components among domains
- There are meta-information-based related works. OSGi R4 and Georgiadis [11] both uses meta-information to automatically connect components.
- None of these work in hierarchical environments and there is the question of the right set of meta-information
- Hierarchical arrangement is important
  - Because it decreases number of connections therefore reconnection time
  - Because it allows reconnection of entire component networks as single component (e.g. moving data synchronization components as one group)

Dynacomp elements

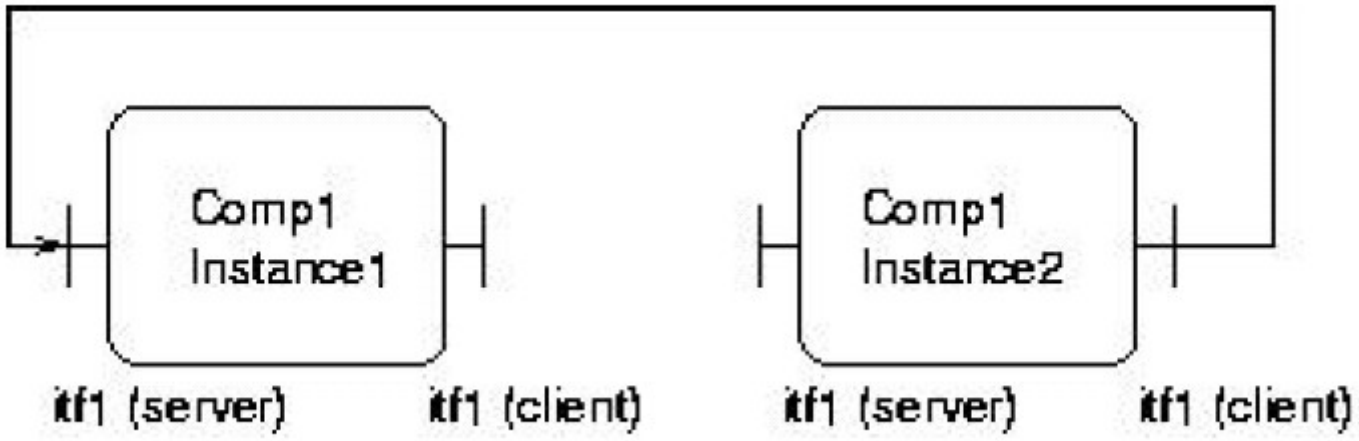
- Dynacomp component types
  - Primitive components: expose meta-information
  - Composite components: arrange components in their domains, move components into and out of the domain according to meta-information
- Meta-information in Dynacomp. Builds on Fractal meta-information but extends it.
  - Interface name.
    - Comes directly from Fractal
  - Interface multiplicity (taken from Service Binder)
    - one, one or more, zero or one, zero or more
  - Priority
    - To resolve ambiguities, the interface pair with the highest sum of priorities will be connected

Meta-information: component and properties

- Properties: way to control component connections. Property is a key-value pair
- Dynacomp assigns primarily properties to components
- It is possible to override component properties with interface properties. These properties are matched only if that particular interface is connected.
- Property language
  - Difficult complexity/efficiency trade-off! The more complex the filter expression is, the more time reconfiguration takes!
  - Service Binder: LDAP Filter
  - Dynacomp:
    - Required property: the component/interface to connect must have that property with the given value in order for the connection to be made.
    - Optional property: if the component/interface to connect has this property, the value must have the given value in order for the connection to be made
    - Forbidden property: the component/interface to connect is not allowed to have this property with this value
  - Optional property cannot be expressed with OSGi R4 LDAP filters

Meta-information: Interface direction

- Interface direction.
  - Beside input and output ports, Dynacomp defines bidirectional ports.
  - Statement to express: certain component instances of same type have to be connected with each other
  - In Dynacomp, this is expressed as a pair of client/server interfaces that are connected in random direction.



Meta-information

- Superproperties: properties that control which domain the component can be placed into. Superproperties of the component are matched against the properties of the composite component
- Export and import list: list of interfaces that can be exported from and imported into this domain
- Timeout: unconnected components “expire” after a timeout
- Creation time: timestamp when the component was created
- Creation lists: lists containing component names and labels. These are reconnection scenarios for certain situations (e.g. if certain even occurs, start creating components from label “event1”)

Reconnection algorithm in short

- Connect the domain according to constraints
- Put every unconnected component into the storage of the domain
- Propagate the storage downward, into the composite components in this domain, considering superproperties and import lists
- Start the components in this domain in dependency order. If any component becomes unresolved during the starting phase, reconnection restarts (a component may delete itself during start)
- Check for timeout and delete components that are in the storage for too long time
- Propagate the storage upward, to the parent of this component considering export list of this domain and reconnect the parent

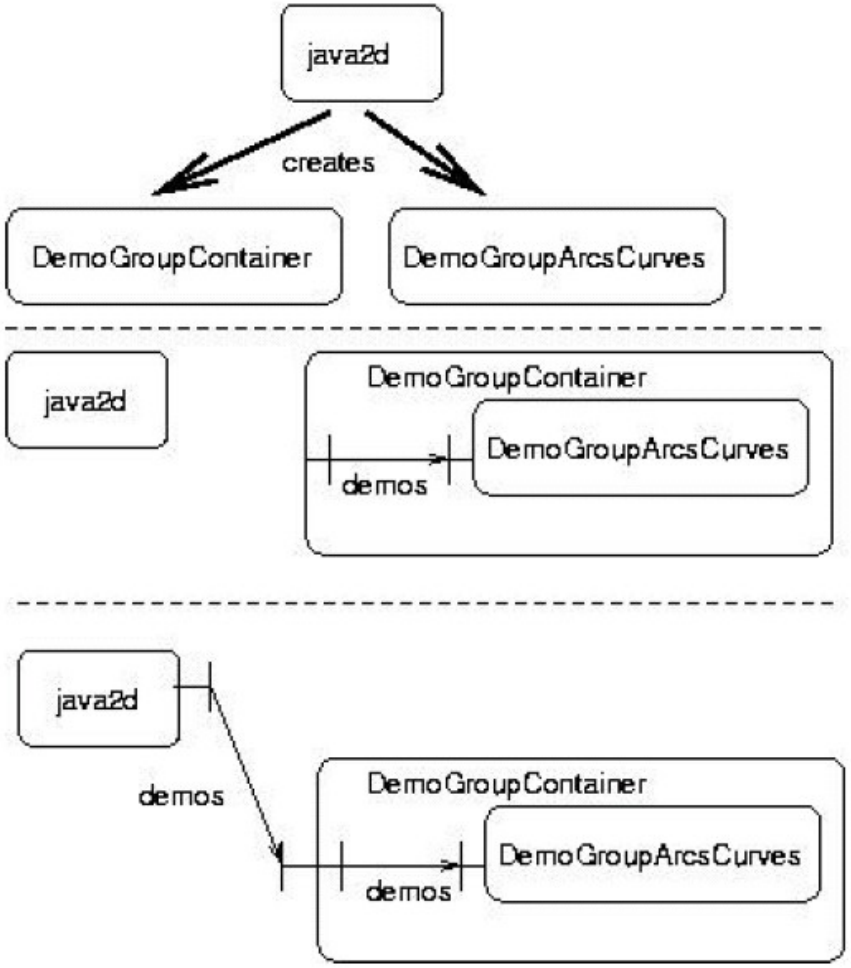
- This algorithm may create endless loops!

Case study: Java2D

- Standard demo shipped with every JDK
- Monolithic application
- Idea: save footprint by componentizing it and activate only the features required by the end user
- Use this mobile-inspired use case to demonstrate Dynacomp
- We will oversee several implemented scenarios

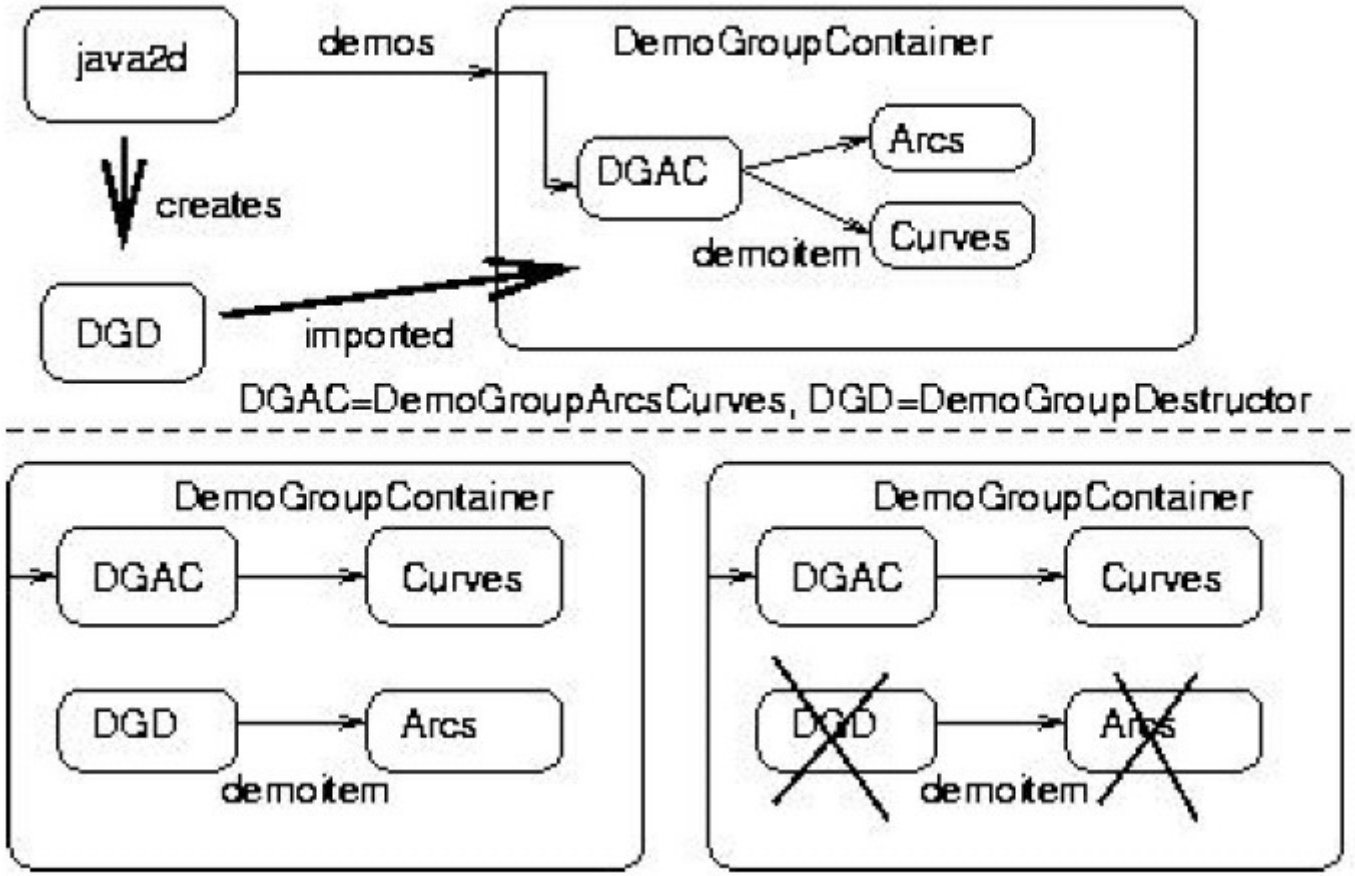
Adding demo groups

- Java2D has demo groups and demos are put into demo groups. One demo group is one tab on the selection panel.



Deleting a demo

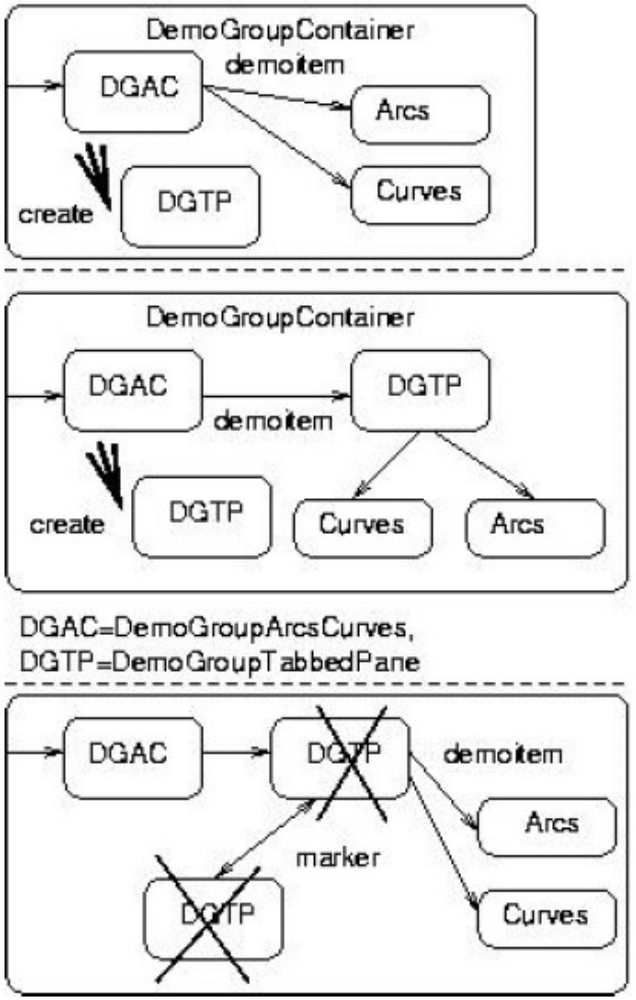
- Destructor component is created which searches and destroys the particular demo. Searching is controlled by properties of the destructor and the demo components



NOKIA
Connecting People

Switching between grid and tabbed pane

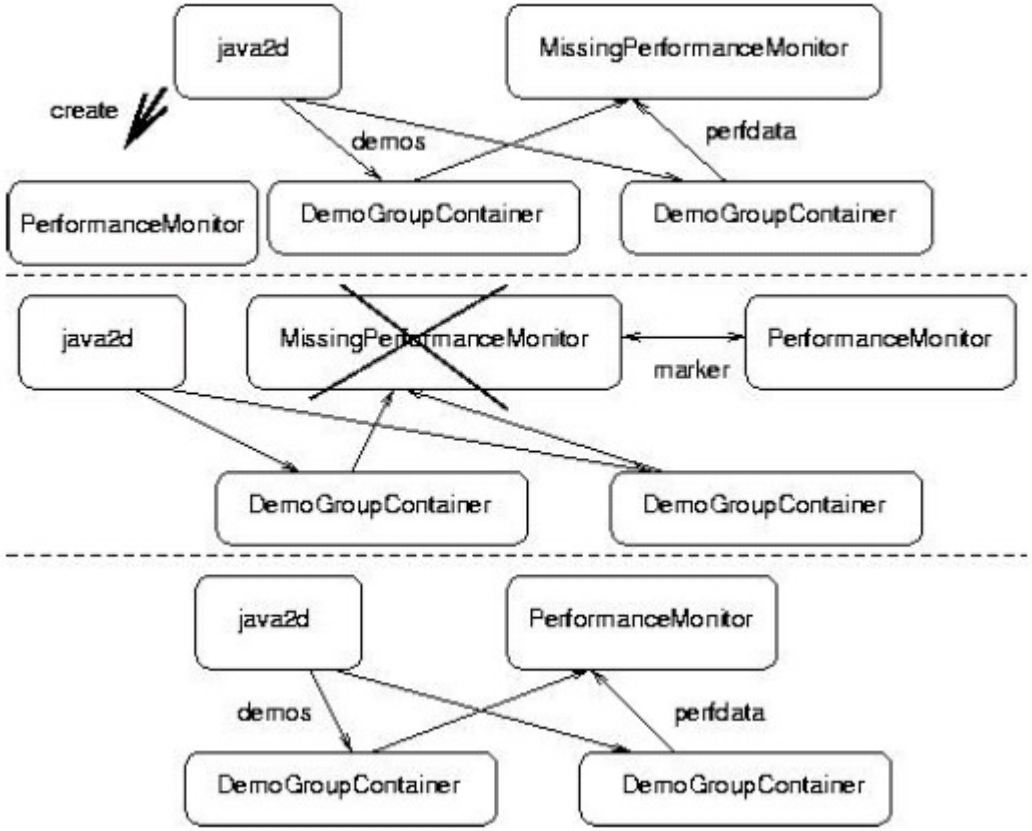
- Demos can be rendered by the demo group as tabbed pane or grid. This is implemented by adding a tabbed pane component to the component network. If this component recognizes that there's already another tabbed pane component in the domain, it destructs both therefore restoring grid view.



NOKIA
Connecting People

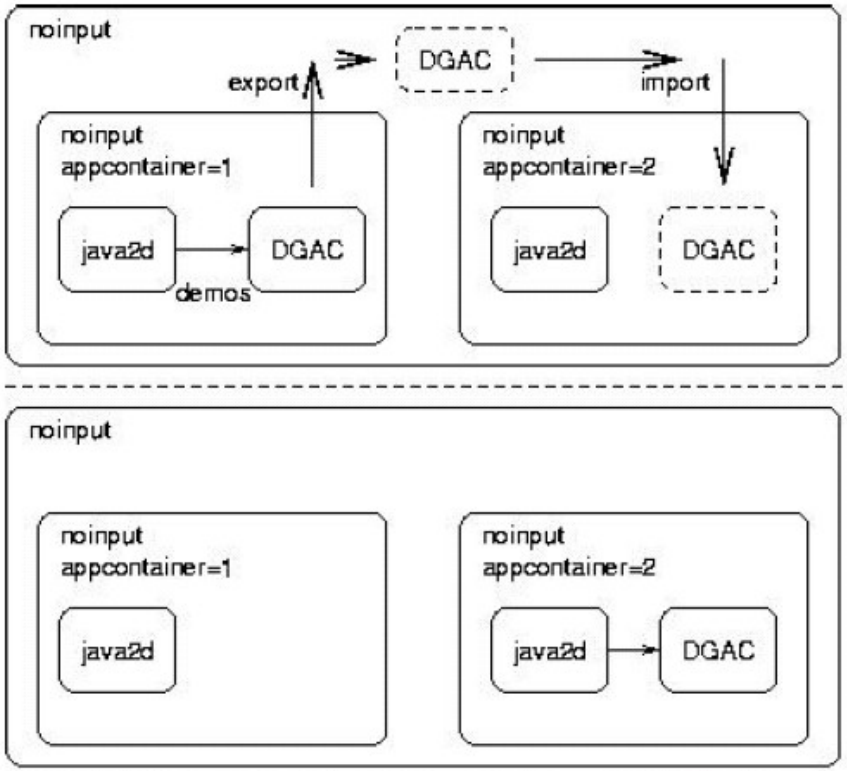
Adding performance monitor

- Performance monitor can be selectively switched on and off. This is accomplished by switching between the dummy and real component.



Moving demo groups between demo frames

- New feature: whole demo groups can be placed to secondary frame



DGAC=DemoGroupArcsCurves (more exactly DemoGroupContainer containing the components belonging to the Arcs&Curves demo group)

NOKIA
Connecting People

Implementation and results

- Download the stuff: http://javasite.bme.hu/~paller/common/dynacomp.tar.gz
- Static footprint (class file size):
  - Java2D's original version: 387964 bytes
  - Dynacomp version: 376769 bytes.
  - Dynacomp framework (including Fractal API and Julia) is 458825 bytes (397729 bytes from Fractal API+Julia and 61096 bytes from the dynamic reconnection facility)
  - With the framework added, Dynacomp version is larger but the framework can be shared by multiple applications

Dynamic footprint

- Orig: original Java2D
- DC: Dynacomp version with only the particular feature activated
- DCAll: Dynacomp version with all the features activated but the particular feature selected

Demo	Orig (kB)	DC (kB)	DCall (kB)
Arcs&Curves	6135	3675	7305
Clipping	6081	4130	7302
Colors	6339	3904	7617
Composite	6309	4149	7854

Blackout

- Time needed to reconfigure the component framework
- During this time, the application is not accessible

Activity	Time (msec)
Inserting an empty demo group as first demo group	361
Inserting a demo into an empty demo group	736
Inserting a demo as the 4th demo in the group	471
Inserting 4 demo groups with a total of 16 demos	6007
Switching from grid to tabbed view	418
Switching from tabbed to grid view	497

Conclusions

- Dynamic adaptation surfaces in many areas: autonomous systems, self-healing, reflective middleware
- Centralized reconfiguration managers are hard to analyze and extend: each new context element to support adds complexity to potentially all the reconfiguration managers. This paper proposed a more distributed approach where general reconfiguration manager does the reconnection based on meta-information.
- Large component networks need hierarchical component system to minimize reconnection time: this is not supported by OSGi R4
- Blackout effect exists during dynamic reconfiguration
- With appropriate framework, dynamic footprint can be significantly decreased
- Julia's static footprint is large