A Survey of Self-Management in Dynamic Software Architecture Specifications

Jeremy S. Bradbury, James R. Cordy[†], Juergen Dingel

Software Technology Laboratory
School of Computing
Queen's University
Kingston, Canada

Michel Wermelinger[‡]

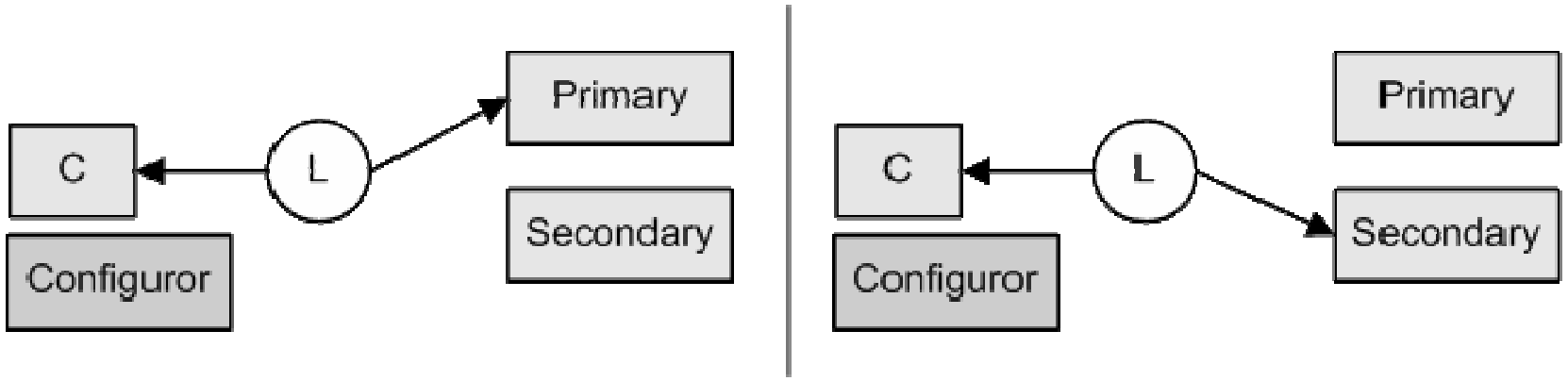
Departamento de Informática
Universidade Nova de Lisboa
Caparica, Portugal

[†] Current address: ITC-IRST, Trento, Italy.
[‡] Current address: Computing Department, The Open University, Milton Keynes, UK.

SUPPORTED BY NSERC CRSNG

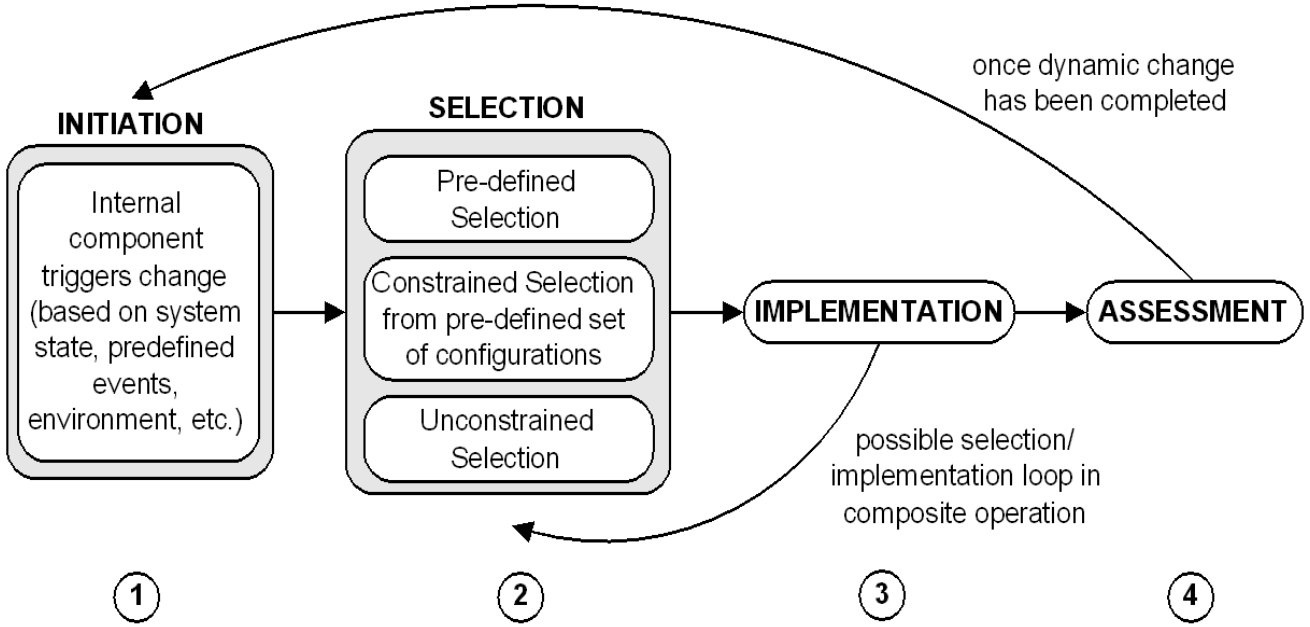
Dynamic Software Architectures

- *Dynamic software architectures* are those architectures that modify their architecture and enact (implement) the modifications during the system's execution.
- Client-Server example:



Self-Managing Architectures

- A *self-managing architecture* not only implements the change internally but also initiates, selects, and assesses the change itself without the assistance of an external user.



Self-Managing Architectures Challenges

- Dynamic components are “challenging in terms of correctness, robustness, and efficiency” [Szy03].
- Self-managing architectures are especially challenging because the initiation and selection also occur internally.
- How can we address these challenges?
  - Formal specification

[Szy03] Clemens Szyperski. Component technology: what, where, and how? In *Proceedings of the 25th International Conference on Software Engineering*, pages 684–693, 2003.

		Architectural Structure		Architectural Element Behavior		Architectural Reconfiguration
		Architectural Style	System Architecture	Components	Connectors	
Graph	Le Métayer approach	context-free graph grammar	graph (formally defined as a multiset)	nodes of a graph and a CSP like behavior specification	edges of a graph	graph rewriting rules with side conditions to refer to the status of public variables
	Hirsch et al. approach	context-free graph grammar	hypergraph	edges of a graph with CCS labels	nodes of a graph [point-to-point communication (white nodes) and broadcast communication (black nodes)]	graph rewriting rules
	Taentzer et al. approach	-	distributed graph (network graph)	local graph for each network graph node and local transformations between local graphs	edges of a graph	graph rewriting rules
	COMMUNITY	-	Categorical diagram	a program in COMMUNITY (a UNITY-like language)	a star-shaped configuration of programs	category theory using double-pushout graph transformation approach
	CHAM	creation CHAM	-	molecule	links between two component molecules	evolution CHAM reaction rules
Process Algebra	Dynamic Wright	-	implicit graph representation (components and connectors are nodes)	ports (interface) + computation (behavior)	roles (interface) + glue (behavior)	CSP
	Darwin	-	implicit graph representation	programming language + component specification of comm. objects	support for simple bindings	π-calculus
	LEDA	-	implicit graph representation	interface specification, composition and attachment specification (if composite)	attachments at top level components	π-calculus
	PiLar	-	implicit graph representation	components with ports, instances of other components and constraints	support for simple bindings	CCS
Logic	Gerel	-	implicit graph representation	interface in Gerel language and behavior in a programming language	defined by bind operation in configuration components	first order logic
	Aguirre-Maibaum approach	-	implicit graph representation	class with attributes, actions and read variables	association consisting of participants and synchronization connections	first order logic, temporal logic
	ZCL	-	implicit graph representation (defined by set of state schemas in Z)	state schema in Z	connection between ports of components	operation schema in Z (predicate logic and set theory)
Other	C2SADEL	only supports the C2 arch. style	implicit graph representation (defined by Architecture Description Language (ADL))	element with top and bottom interface and behavior (defined by Interface Definition Language (IDL))	element with top and bottom ports and filtering mechanisms (defined by ADL)	Architecture Modification Language (AML)
	RAPIDE	-	implicit graph representation	types language for component interface (plus other sublanguages for behavior)	broadcast connection rule (||>) or pipe (=>)	where statement or execution architecture events

Purpose of Survey

- We have 3 criteria for our survey:
  1. Determine if each approach supports self-management.
  2. Evaluate each approach with respect to expressiveness.
     a) Types of change supported.
     b) Types of selection mechanisms supported.
  3. Evaluate each approach with respect to the distribution (scalability) of the management approach.

Support for Self Management

- We evaluate support for self-management by determining if initiation occurs internally.
- In all of the approaches we evaluated, if initiation occurs internally then the other steps of the change process can also be specified internally.

Support for Self Management

		Initiation	
		Internal	External
Graph	Le Métayer approach	●	●
	Hirsch et al. approach	○	○
	Taentzer et al. approach	?	?
	COMMUNITY	●	●
	CHAM	●	●
Process Algebra	Dynamic Wright	●	○
	Darwin	●	○
	LEDA	●	○
	PiLar	●	○
Logic	Gerel	●	◉
	Aguirre-Maibaum approach	●	○
	ZCL	●	○
Other	C2SADEL	○	◉
	RAPIDE	●	○

KEY:

- ● = supported by specification explicitly
- ◉ = supported externally (i.e. tool, language, etc)
- ○ = not supported
- ? = support unknown

Support for Self Management

		Initiation	
		Internal	External
Graph	Le Métayer approach	●	●
	Hirsch et al. approach	○	○
	Taentzer et al. approach	?	?
	COMMUNITY	●	●
	CHAM	●	●
Process Algebra	Dynamic Wright	●	○
	Darwin	●	○
	LEDA	●	○
	PiLar	●	○
Logic	Gerel	●	◉
	Aguirre-Maibaum approach	●	○
	ZCL	●	○
Other	C2SADEL	○	◉
	RAPIDE	●	○

Le Métayer uses side conditions in rewriting rules to refer to public variables in components.

e.g. {Component(c), *c.leave=true*, Connector(c,m), Connector(m,c)}
$\rightarrow \emptyset$

Expressiveness (Types of Change)

- Self-managing architectures are limited by the reconfiguration operations that are available.
  - Example:
    - System that can add connectors but not components is limited – less types of change are available.

Expressiveness (Types of Change)

- We survey support for:
  - Basic operations.
    - component addition and removal.
    - connector addition and removal.
  - Composite operations.
    - Subsystem addition and removal.
    - Available constructs that can be used in specifying composite operation (e.g. sequencing, choice, and iteration).

Expressiveness (Types of Change)

		Basic Reconfiguration Operations				Composite Operations			
							Operation Constructs		
		Component Addition	Component Removal	Connector Addition	Connector Removal	Basic Support	Sequencing	Choice	Iteration
Graph	Le Métayer approach	●	●	●	●	●	○	●	○
	CommUnity	●	●	●	●	●	●	●	●
	CHAM	●	●	●	●	●	i	i	i
Process Algebra	Dynamic Wright	●	●	●	●	●	●	●	●
	Darwin	●	⦿	–	–	●	⦿?	⦿?	⦿?
	LEDA	●	○	●	○	○	○	●	○
	PiLar	●	●	●	●	●	●	●	○
Logic	Gerel	●	●	●	●	●	●	●	●
	Aguirre-Maibaum approach	●	●	●	●	●	?	●	?
	ZCL	●	●	●	●	●	?	?	?
Other	Rapide	●	●	●	●	●	?	●	○

KEY:

- ● = supported by specification explicitly
- i = supported by specification implicitly
- ⦿ = supported externally (i.e. tool, language, etc)
- ○ = not supported
- ? = support unknown
- – = not applicable

Expressiveness (Types of Change)

		Basic Reconfiguration Operations				Composite Operations			
							Operation Constructs		
		Component Addition	Component Removal	Connector Addition	Connector Removal	Basic Support	Sequencing	Choice	Iteration
Graph	Le Métayer approach	●	●	●	●	●	○	●	○
	CommUnity	●	●	●	●	●	●	●	●
	CHAM	●	●	●	●	●	i	i	i
Process Algebra	Dynamic Wright	●	●	●	●	●	●	●	●
	Darwin	●	⊙	–	–	●	⊙?	⊙?	⊙?
	LEDA	●	○	●	○	○	○	●	○
	PiLar	●	●	●	●	●	●	●	○
Logic	Gerel	●	●	●	●	●	●	●	●
	Aguirre-Maibaum approach	●	●	●	●	●	?	●	?
	ZCL	●	●	●	●	●	?	?	?
Other	Rapide	●	●	●	●	●	?	●	○

RAPIDE has one execution architecture event type for each basic operation:

CreateModule(...);
DeleteModule(module : Event);
CreatePathway(...);
DeletePathway(pathway : Event);

Expressiveness (Types of Change)

		Basic Reconfiguration Operations				Composite Operations			
							Operation Constructs		
		Component Addition	Component Removal	Connector Addition	Connector Removal	Basic Support	Sequencing	Choice	Iteration
Graph	Le Métayer approach	●	●	●	●	●	○	●	○
	CommUnity	●	●	●	●	●	●	●	●
	CHAM	●	●	●	●	●	i	i	i
Process Algebra	Dynamic Wright	●	●	●	●	●	●	●	●
	Darwin	●	⊙	–	–	●	⊙?	⊙?	⊙?
	LEDA	●	○	●	○	○	○	●	○
	PiLar	●	●	●	●	●	●	●	○
Logic	Gerel	●	●	●	●	●	●	●	●
	Aguirre-Maibaum approach	●	●	●	●	●	?	●	?
	ZCL	●	●	●	●	●	?	?	?
Other	Rapide	●	●	●	●	●	?	●	○

Example of iteration in CommUnity approach:

```
script RestoreStandard
  in a: Account
  prv i: record(c:Customer; co:VIP)
  for i in match {c:Customer: co:VIP | co(c,a)}loop
     remove i.co;
     create standard(i.c, a);
  end loop
end script
```

Expressiveness (Types of Selection)

- We distinguish between 3 levels of selection that a specification approach may support:
  - *Pre-defined Selection:* A change is chosen based on a pre-defined selection made prior to run-time.
  - *Constrained Selection from a Pre-defined Set:* There is some choice in what change to make.
  - *Operationally Unconstrained Selection:* There is an unconstrained choice regarding the change to make.

Expressiveness (Types of Selection)

		Selection		
		Pre-defined	Constrained from Pre-defined set	Unconstrained
Graph	Le Métayer approach	●	●	○
	COMMUNITY	●	●	○
	CHAM	●	●	○
Process Algebra	Dynamic Wright	●	●	○
	Darwin	●	○?	○
	LEDA	●	●	○
	PiLar	●	○	○
Logic	Gerel	●	?	○
	Aguirre-Maibaum approach	●	●?	○
	ZCL	●	○	○
Other	RAPIDE	●	●	○

KEY:

● = supported by specification explicitly

○ = not supported

? = support unknown

Expressiveness (Types of Selection)

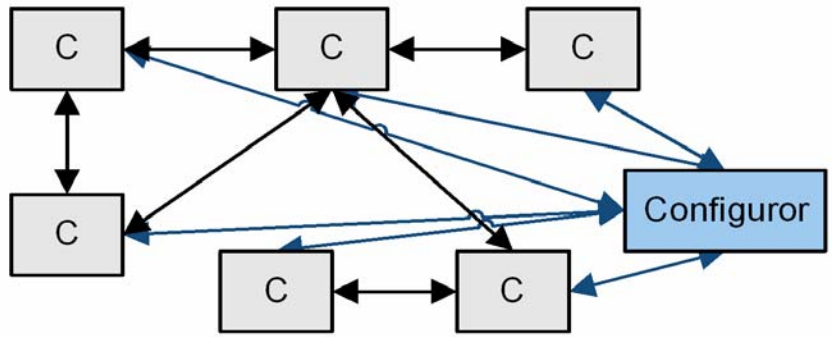
		Selection		
		Pre-defined	Constrained from Pre-defined set	Unconstrained
Graph	Le Métayer approach	●	●	○
	COMMUNITY	●	●	○
	CHAM	●	●	○
Process Algebra	Dynamic Wright	●	●	○
	Darwin	●	○?	○
	LEDA	●	●	○
	PiLar	●	○	○
Logic	Gerel	●	?	○
	Aguirre-Maibaum approach	●	●?	○
	ZCL	●	○	○
Other	RAPIDE	●	●	○

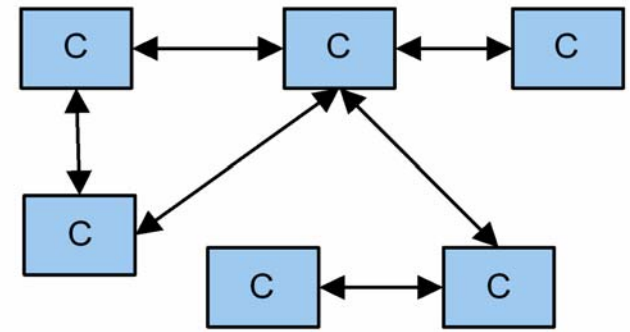
Example of constrained choice between two component attachments:

```
DynamicClientServer {
 interface none;
 composition
    client: Client;
    server[2]: Server;
 attachments
    client request(r)<>   binds
       if (server[1].n <= server[2].n)
          then server[1].serve(r);
          else server[2].serve(r);
}
```

Distribution (Scalability)

- The management of reconfiguration in dynamic software architectures can be either:
  - Centralized in a specialized component
  - Distributed across components





Distribution (Scalability)

		Management	
		Centralized	Distributed
Graph	Le Métayer approach	●	○
	COMMUNITY	●	○
	CHAM	●	○
Process Algebra	Dynamic Wright	●	○
	Darwin	●	●?
	LEDA	●	●?
	PiLar	○	●
Logic	Gerel	◉	○
	Aguirre-Maibaum approach	●	○
	ZCL	●	○
Other	RAPIDE	●	●

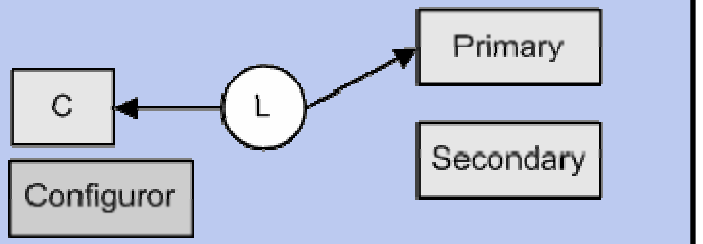
KEY:

- ● = supported by specification explicitly
- ◉ = supported externally (i.e. tool, language, etc)
- ○ = not supported
- ? = support unknown

Distribution (Scalability)

		Management	
		Centralized	Distributed
Graph	Le Métayer approach	●	○
	CommUnity	●	○
	CHAM	●	○
Process Algebra	Dynamic Wright	●	○
	Darwin	●	●?
	LEDA	●	●?
	PiLar	○	●
Logic	Gerel	◉	○
	Aguirre-Maibaum approach	●	○
	ZCL	●	○
Other	Rapide	●	●

In Dynamic Wright reconfigurations are specified in a configuror.



Distribution (Scalability)

		Management	
		Centralized	Distributed
Graph	Le Métayer approach	●	○
	CommUnity	●	○
	CHAM	●	○
Process Algebra	Dynamic Wright	●	○
	Darwin	●	●?
	LEDA	●	●?
	PiLar	○	●
Logic	Gerel	◉	○
	Aguirre-Maibaum approach	●	○
	ZCL	●	○
Other	Rapide	●	●

In the PiLar language multiple components can have constraints that specify localized reconfigurations.

Conclusions

- Most approaches support basic self-management.
  - Support specification and analysis related to the implementation of change.
- Areas that need to be addressed further include:
  - Initiation (using monitors, etc.)
  - Selection of change (Expressiveness).
  - Distributed management (Scalability).
  - Analysis of entire architectural change process.

Future Work & Open Questions

- What other criteria should be used for the assessment of specifications for self-managed architectures (in addition to basic support, expressiveness, and distribution)?
- In terms of analysis, what questions need to be addressed?
- Should a benchmark for specifications be developed? Possibly based on practical examples?
- In general, is specification work in this area going in the right direction?
  - There seems to be a lack of practical use of a number of the approaches surveyed…

A Survey of Self-Management in Dynamic Software Architecture Specifications

Jeremy S. Bradbury, James R. Cordy[†], Juergen Dingel

Software Technology Laboratory
School of Computing
Queen's University
Kingston, Canada

Michel Wermelinger[‡]

Departamento de Informática
Universidade Nova de Lisboa
Caparica, Portugal

[†] Current address: ITC-IRST, Trento, Italy.
[‡] Current address: Computing Department, The Open University, Milton Keynes, UK.

SUPPORTED BY NSERC CRSNG