On the reconfiguration of software connectors [*]

Nuno Oliveira[†]
HASLab / INESC TEC
Universidade do Minho
Braga, Portugal
nunooliveira@di.uminho.pt

Luís S. Barbosa
HASLab / INESC TEC
Universidade do Minho
Braga, Portugal
lsb@di.uminho.pt

ABSTRACT

Software connectors encapsulate interaction patterns between services in complex, distributed service-oriented applications. Such patterns evolve over time, in response to faults, changes in the expected QoS levels, emergent requirements or the reassessment of contextual conditions. This paper builds up on a model for connector reconfiguration to introduce notions of reconfiguration equivalence and refinement allowing for reasoning about them. This paves the way towards a (still missing) calculus of connector reconfigurations.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Architectural reconfiguration

Keywords

Software architecture, Software connectors, Reconfiguration

1. INTRODUCTION

Complex distributed service-oriented systems require reliable and yet flexible architectures. A clear separation between services/components and the protocols that manage their interaction seems to be a step in the right direction towards compositional design, analysis and verification. Exogenous coordination models, based on software connectors, such as Reo [1], offer powerful "glue-code" to express such interaction protocols, while maintaining the envisaged separation of concerns.

These systems are subject to constant evolution, entailing the need for some kind of dynamic reconfiguration of their interaction protocols [12]. Such needs may be due to changes in either requirements or in runtime contextual conditions which often degrades expected QoS values to unacceptable levels.

Conventionally an architectural reconfiguration mainly targets the manipulation of components [13, 21]. Alternatively to this high-level stand point, a reconfiguration may target the interaction protocols, *i.e.,* the connector's structure. Such reconfigurations substitute, add or remove communication channels, move communication interfaces between components, and may even restructure a complex interaction policy. Connector reconfiguration mechanisms play, in this setting, a major role to express change and adaptation in interaction protocols.

Reference [20] introduced a model for such mechanisms in which connectors are 'syntactically' represented by a graph of *communication channels*, whose nodes stand for interaction points and edges are labelled with channel identifiers and types defining their behaviour. Such graphs are referred to as *coordination patterns.* The model defines a number of elementary reconfiguration primitives, which, on their turn, are combined to yield 'big-step' reconfiguration patterns to manipulate significative parts of an architecture.

The present paper builds on this model to discuss criteria of assessing and comparing reconfigurations. In particular, two classes of criteria are proposed. One is *structural* and independent of the actual semantics of coordination patterns. It is used, for example, to require that along a reconfiguration a specific type of channel remains attached to a given end or set of ends. Such properties are specified in a propositional hybrid logic interpreted over the coordination pattern. The second criterium, on the other hand, resorts to the specific semantics chosen for the coordination patterns. It may be used, for example, to require that a reconfiguration preserves the overall interaction *behaviour* or, at least, part of it. Comparing reconfigurations along these different criteria constitutes the main contribution of this paper. Coordination patterns considered in this paper are framed into the Reo coordination model. The proposed criteria, however, are still valid in other coordination models, as long as they can be 'syntactically' represented by a graph (of communicating devices).

Outline. Section 2 sums up the proposed framework for

[*]This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-010047.
[†]This author is supported by an Individual Doctoral Grant from FCT, with reference SFRH/BD/71475/2010.

reconfiguration of software connectors, based on a number of elementary operations which are combined to yield 'big-step' reconfiguration patterns. The two following sections introduce mechanisms for assessing and comparing reconfigurations from two orthogonal perspectives: *behavioural* (section 3), resorting to whichever semantic model is chosen for the underlying coordination model, and *structural* (section 4), in which properties of channel interconnection are expressed in a variant of propositional hybrid logic. Section 6 concludes the paper after a review of related work in Section 5.

2. RECONFIGURATION MECHANISMS

This section provides a primer on the underlying reconfiguration framework, as a background for what follows. A comprehensive discussion can be found in [20].

2.1 Coordination Patterns

Software connectors, in the context of this research, correspond to coordination patterns encoding reusable solutions for architectural problems in distributed, loosely-coupled systems. Formally, a coordination pattern is given as a graph of *channels* whose nodes represent interaction points and edges are labelled with channel identifiers and types. To provide a concrete illustration of this approach, the Reo framework [3, 1] is adopted.

Henceforth a channel is considered as in Reo: a point-to-point (abstract) communication device with a unique identifier, a behaviour (or coordination protocol) and two ends. It allows for data flow by accepting it on its *source* end and dispensing from the *sink* end. Each channel has exactly two ends and are, normally, directed (with a source and a sink end) but Reo also accepts undirected channels (*i.e.,* channels with two ends of the same sort). Figure 1 recalls the basic types of channels in Reo.

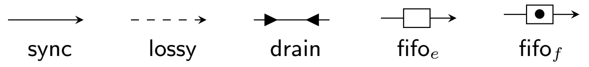


Figure 1: Primitive Reo channels.

The sync channel transmits data from one end to another whenever there is a request at both ends synchronously, otherwise one request shall wait for the other. The lossy channel behaves likewise, but data may be lost whenever a request at the source end is not corresponded by another at the sink end. Differently, a fifo channel has a buffering capacity of one memory position, therefore allowing for asynchronous occurrence of I/O requests. Qualifier e or f refers to the channel internal state (either *empty* or *full*, respectively). Finally, the synchronous drain channel accepts data synchronously at both ends, losing it.

Channel ends form nodes which can be connected to assembly more complex connectors. Nodes may be of three distinct types: (*i*) source node, if it connects only source channel ends; (*ii*) sink node, if it connects only sink channel ends and (*iii*) mixed node, if it connects both source and sink nodes. Source and sink nodes are also referred to as the *boundary nodes* of a connector. As an example of channel composition, Figure 2 depicts a *sequencer* connector which results from the composition of several channels.

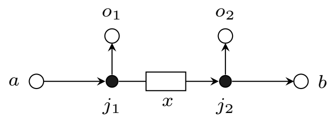


Figure 2: The *sequencer*. A composition of five sync channels and one fifo$_e$ channel. Graphically, white circles mean both source and sink nodes used to link the connector to services, while black ones mean mixed (internal) nodes.

Let $\mathcal{N} \cup \{\bot\}$ and $\mathcal{I}$ denote, respectively, a set of nodes and a set of channel identifiers. $\bot$ is used to represent a node which is neither a source nor a sink node. Moreover, let $\mathcal{T}$ stand for the set of primitive channel types in Reo. The connector's assembly structure is represented by a coordination pattern defined as follows.

Definition 1 (Coordination pattern). *A coordination pattern, ρ, is a graph on connector ends whose edges are labelled with instances of primitive channels, represented by a channel identifier, $id \in \mathcal{I}$, and a type $t \in \mathcal{T}$, formally defined as*

$$\rho \overset{\text{def}}{=} R \subseteq \mathcal{N} \times \mathcal{I} \times \mathcal{T} \times \mathcal{N}$$

Operations $I(\rho)$ and $O(\rho)$ are used to retrieve, respectively, the set of source and sink nodes from coordination pattern ρ. Optionally, sets I and O for source and sink nodes, respectively, may be added to the pattern structure for clarity (as shown in the two examples below). The set of all coordination patterns is denoted by $\mathcal{P}$.

Clearly, every channel instance gives rise to a coordination pattern. For example

$$\langle \{a\}, \{b\}, \{\langle a, sc, \mathsf{sync}, b\rangle\}\rangle$$

corresponds to a single sync channel, identified by sc, linking an input port a to an output port b. Similarly, plugging to its output port a drain channel yields a dummy synchroniser which allows data to be written on a, if there exist pending requests at c

$$\langle \{a, c\}, \emptyset, \{\langle a, sc, \mathsf{sync}, b\rangle, \langle b, dr, \mathsf{drain}, \bot\rangle, \langle c, dr, \mathsf{drain}, \bot\rangle\}\rangle.$$

A drain has two source ends. Therefore, a pattern formed by an instance of a drain channel resorts to the special end $\bot$, which intuitively represents a *hole* where data is lost, meaning absence of data flow.

In the sequel, the visual Reo-like representation of coordination patterns (as in Figure 2) is used as an abbreviation of the formal model.

2.2 Reconfigurations

A reconfiguration is defined as a non-empty sequence of elementary operations that manipulate the internals of a coordination pattern. Set $\mathcal{O} \overset{\text{def}}{=} \{\mathsf{par}, \mathsf{join}, \mathsf{split}, \mathsf{remove}\}$ is the set of such elementary operations, which are described below. The application of a reconfiguration r to $\rho \in \mathcal{P}$ yields a new coordination pattern and is denoted by $\rho \bullet r$.

par: $\rho \bullet \mathsf{par}(\rho')$ sets $\rho, \rho' \in \mathcal{P}$ in parallel without creating any connection between them. The par operation assumes, without loss of generality, that both the nodes and channel identifiers in the patterns to be joined are disjoint.

join: $\rho \bullet \mathsf{join}(P, j)$ creates a new node $j \in \mathcal{N}$ that superposes all nodes in a given set $P \subseteq \mathcal{N}$. The join operation has two pre-conditions. Clearly, j must be a fresh name in $\rho \in \mathcal{P}$, unless it is in P. Additionally, every node in P shall exist as a node of ρ.

split: $\rho \bullet \mathsf{split}(p)$ is dual to join. It takes a node $p \in \mathcal{N}$ in $\rho \in \mathcal{P}$ and breaks connections, separating all channel ends coincident in p. Technically this is achieved by renaming every occurrence of node p in ρ to a fresh name.

remove: $\rho \bullet \mathsf{remove}(id)$ removes a channel, identified by $id \in \mathcal{I}$, from $\rho \in \mathcal{P}$, if it exists.

One may ask whether the application of these operations upon a coordination pattern, can leave isolated nodes. However, it can be proved (by induction on the expression representing the reconfiguration script) this never occurs. For space limitations, the formal specifications of the reconfiguration operations and their properties are omitted. The interested reader is referred to [20], for a detailed account on the omitted semantics.

2.3 Reconfiguration patterns

The focus of traditional reconfiguration in software architecture [21] is the replacement of individual components, rather than the transformation of the underlying interaction protocols. The present approach goes in the other direction. However, still at this level, the interest is in defining 'big step' reconfigurations, referred to as reconfiguration patterns, regarding them as sequences of elementary reconfigurations, which affect significant parts of a connector (rather than just a point or a channel), and are generic and reusable. Figure 3 shows a (not closed) set of reconfiguration patterns that are useful in practice. The following paragraphs describe briefly the purpose of each of them.

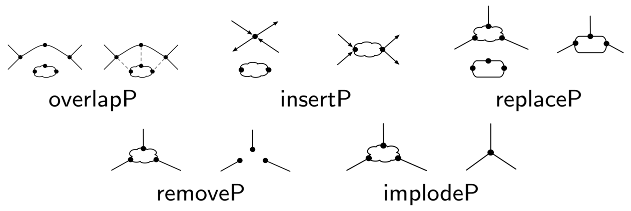


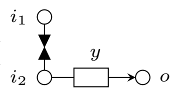
Figure 3: Reconfiguration patterns.

removeP: $\rho \bullet \mathsf{removeP}(C)$ takes a set $C \subseteq \mathcal{I}$ of channel identifiers and removes them from $\rho \in \mathcal{P}$, by successive application of elementary remove operations.

overlapP: $\rho \bullet \mathsf{overlapP}(\rho_r, T)$ connects a new pattern $\rho_r \in \mathcal{P}$ to $\rho \in \mathcal{P}$ by joining specific nodes in $T \subseteq \mathcal{N} \times \mathcal{N} \times \mathcal{N}$. Each triple in T indicates which pairs of nodes from ρ and ρ_r are to be joined (overlapped) and which name is given to the result.

insertP: $\rho \bullet \mathsf{insertP}(\rho_r, n, m_i, m_o, j_i, j_o)$ puts $\rho, \rho_r \in \mathcal{P}$ side by side and splits $n \in \mathcal{N}$ of ρ to make room for the new coordination pattern to be added. Connections are then re-built as follows: all the output ports produced by the split operation are joined with $m_i \in \mathcal{N}$ producing a new node $j_i \in \mathcal{N}$. Dually, the input ports produced by the split operation are joined with $m_o \in \mathcal{N}$ resulting in a new node $j_o \in \mathcal{N}$.

Example. Consider the *sequencer* coordination pattern in Figure 2. Suppose that a company uses this protocol to coordinate the sequential execution of two services coupled to ports o_1 and o_2. For some reason, there was a need to restrict the second service to execute only after the first one finishes. A possible solution is to let the first service to acknowledge its termination and the protocol to memorise it. If ρ_s is the *sequencer* coordination pattern one may propose the following reconfiguration

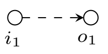
$r_{proactive} = \rho_s \bullet \mathsf{insertP}($ 

$, j_2,\ i_2,\ o,\ k_1,\ k_2)$

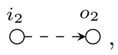
which yields the *proactive waiting sequencer* coordination pattern presented in Figure 4.

replaceP: $\rho \bullet \mathsf{replaceP}(\rho_r, T, C)$ replaces a sub-structure of $\rho \in \mathcal{P}$ by removing the old structure composed of the channels in $C \subseteq \mathcal{I}$ and overlapping ρ_r via information in set $T \subseteq \mathcal{N} \times \mathcal{N} \times \mathcal{N}$.

Example. Consider again the *sequencer* coordination pattern. Imagine now that the services coupled to ports o_1 and o_2 may fail for long periods of time. A deadlock problem could arise if this happens. A possible solution for such problem is not to enforce the services to answer when off. Replacing the sync channels that provide ports o_1 and o_2 by lossy channels could solves the problem. Reconfiguration

$r_{weak} = \rho_s \bullet \langle \mathsf{replaceP}($ 

$, \{(j_1, i_1, k_1), (o_1, o_1, o_1)\}),$

$\mathsf{replaceP}($ 

$, \{(j_2, i_2, k_2), (o_2, o_2, o_2)\})\rangle$

would transform the *sequencer* coordination pattern to a configuration which avoids deadlock, *cf.*, the *weak sequencer* coordination pattern in Figure 4.

implodeP: $\rho \bullet \mathsf{implodeP}(X, C, j)$ collapses a sub-structure of $\rho \in \mathcal{P}$ delimited by the nodes in set $X \subseteq \mathcal{N}$ and composed of the channels in $C \subseteq \mathcal{I}$. The resulting ends are joined together into a new node $j \in \mathcal{N}$.

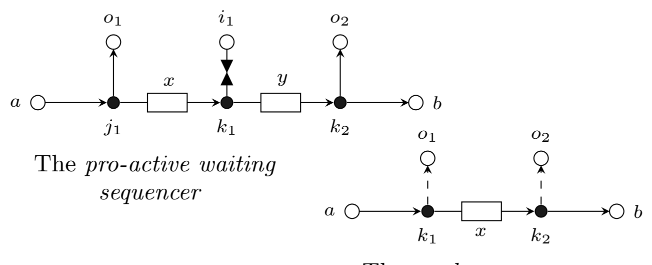


Figure 4: Reconfigurations of the *sequencer* pattern.

A formal account of these reconfiguration patterns is given in [20].

3. REASONING ABOUT RECONFIGURATIONS: BEHAVIOUR

The remainder of this paper investigates criteria to order reconfigurations and enable the working software architect to choose among them. The objective is to endow the architect with means to rule out configurations that, *e.g.,* fail to preserve the overall interconnection behaviour or introduce channels which are, for some reason, not immediately available. This section introduces a *behaviour-based* criterium for reasoning about reconfigurations. The discussion on a complementary *structural* criterium is left for Section 4. The behavioural point of view requires a concrete semantic model for the coordination pattern. For illustration purposes, models of Reo—*constraint automata* (CA) [6, 5] and *reo automata* (RA) [9]—are adopted. The reader is referred to Appendix A, where the definitions of these automata are reproduced. It shall be made clear, however, that the same principle would apply to different semantic models as long as they are amenable to be expressed by a graph of communicating devices.

3.1 Comparing reconfigurations

At this level reconfigurations are compared with respect to the underlying semantic model. In particular, standard notions of similarity and bisimilarity in such models [5, 9] can be used to compare the behaviour of the coordination pattern before and after the application of a reconfiguration. Alternatively, they are used to compare the effect of applying different reconfigurations to the same pattern.

Let $[\![\rho]\!]_{\mathfrak{M}}$ stand for the meaning of the coordination pattern represented by ρ in model $\mathfrak{M}$; $\sim$ and $\preceq$ are, respectively, the bisimilarity and similarity relations in $\mathfrak{M}$. Reference to $\mathfrak{M}$ can be omitted when the model is clear from the context.

Therefore,

Definition 2. *Let $\rho \in \mathcal{P}$, r_1, r_2 be reconfigurations and $\mathfrak{M}$ a semantic model. Then,*

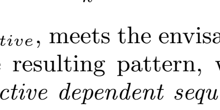
$$r_1 \doteq_{\mathfrak{M}} r_2 \quad \text{iff} \quad \forall_{\rho \in \mathcal{P}} \,.\, [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$$
$$r_1 \preccurlyeq_{\mathfrak{M}} r_2 \quad \text{iff} \quad \forall_{\rho \in \mathcal{P}} \,.\, [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$$

In practice, however, reconfigurations are better compared with respect to their application to a specific coordination pattern, which leads to the following definition,

Definition 3. *Let $\rho \in \mathcal{P}$, r_1, r_2 be reconfigurations and $\mathfrak{M}$ a semantic model. Then,*

$$(r_1 \doteq_{\mathfrak{M}} r_2)_\rho \quad \text{iff} \quad [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$$
$$(r_1 \preccurlyeq_{\mathfrak{M}} r_2)_\rho \quad \text{iff} \quad [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$$

Example. Consider again the *sequencer* coordination pattern. Suppose that a new requirement forces a strict dependence between services in a row. In practice, suppose the second service (coupled to port o_2) is launched with the output of the first service and such an output is memorised whenever the second service is not ready to consume it. Reconfiguration

$r_{dependent} = \mathsf{insertP}($  $, j_2, i_2, o, k_1, k_2)$

which is akin to $r_{proactive}$, meets the envisaged requirement. Figure 5 presents the resulting pattern, which will be referred to as the *pro-active dependent sequencer*. Its RA semantics is exactly the same of the coordination pattern produced by the $r_{proactive}$ reconfiguration. So, although both patterns are slightly different, they exhibit a bisimilar behaviour, expressed in the RA model. Therefore,

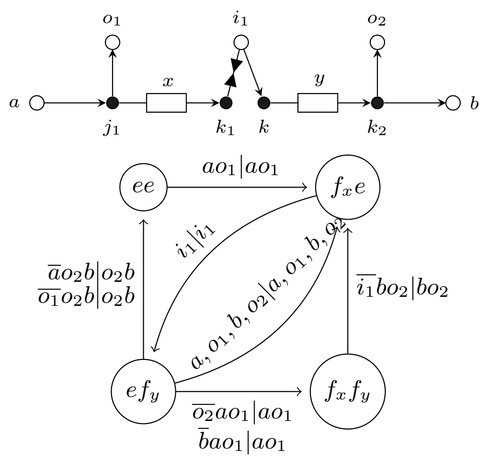


Figure 5: The *pro-active dependent sequencer* coordination pattern and its RA semantics.

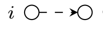
$$(r_{proactive} \doteq_{RA} r_{dependent})_{\rho_s}$$

Note this would not be the case if CA was chosen as a semantic model.

3.2 Refinements and classification

A complete ontology of reconfigurations has to consider fine grained variants of both $\doteq$ and $\preccurlyeq$. For $\doteq$, for example, one may consider whether $[\![\rho]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$, or $[\![\rho]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$, or else $[\![\rho \bullet r_1]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r_2]\!]_{\mathfrak{M}} \preceq [\![\rho]\!]_{\mathfrak{M}}$. For the inequality case, the alternatives are even more: for example, $[\![\rho]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$ or $[\![\rho \bullet r_1]\!]_{\mathfrak{M}} \preceq [\![\rho]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$, among others.

Example. Consider again the *sequencer* coordination pattern. Suppose that the results delivered by the second service are a complement to those offered by the first. Therefore, whenever it fails, the system may proceed normally through port b, disregarding port o_2. This requirement is met by applying the following reconfiguration:

$r_{Qweak} = \rho_s \bullet \mathsf{replaceP}($  $, \{(j_1, i, k_1), (o_2, o_2, o_2)\})$

which is actually part of the reconfiguration r_{weak} discussed before. The resulting coordination pattern (referred to as the *quasi-weak sequencer*) is a variant of the *weak sequencer*. Figure 6 presents its structure along with the semantic models of both coordination patterns.

Clearly, for $\mathfrak{M} \in \{\mathsf{CA}, \mathsf{RA}\}$, $[\![\textit{quasi weak sequencer}]\!]_{\mathfrak{M}} \preceq [\![\textit{weak sequencer}]\!]_{\mathfrak{M}}$. Therefore, $(r_{quasiweak} \preccurlyeq r_{weak})_{\rho_s}$. But one may be more concrete here: both patterns simulate the *sequencer*, leading to

$$[\![\rho_s]\!]_{\mathfrak{M}} \preceq [\![\rho_s \bullet r_{quasiweak}]\!]_{\mathfrak{M}} \preceq [\![\rho_s \bullet r_{weak}]\!]_{\mathfrak{M}}$$

These relations suggest a possible classification of reconfigurations w.r.t. a coordination pattern and a semantic model as follows

Definition 4. *Let $\rho \in \mathcal{P}$, r a reconfiguration, and $\mathfrak{M}$ a semantic model. Then,*

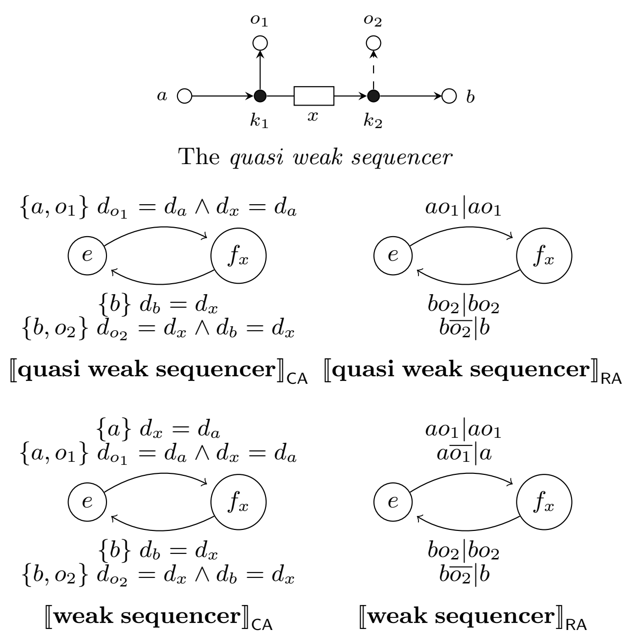

o_1
o_2
a
b
k_1
x
k_2
The *quasi weak sequencer*
$\{a, o_1\}\ d_{o_1} = d_a \wedge d_x = d_a$
e
f_x
$\{b\}\ d_b = d_x$
$\{b, o_2\}\ d_{o_2} = d_x \wedge d_b = d_x$
$ao_1|ao_1$
$bo_2|bo_2$
$b\overline{o_2}|b$
$[\![$quasi weak sequencer$]\!]_{\mathsf{CA}}$
$[\![$quasi weak sequencer$]\!]_{\mathsf{RA}}$
$\{a\}\ d_x = d_a$
$\{a, o_1\}\ d_{o_1} = d_a \wedge d_x = d_a$
$\{b\}\ d_b = d_x$
$\{b, o_2\}\ d_{o_2} = d_x \wedge d_b = d_x$
$ao_1|ao_1$
$a\overline{o_1}|a$
$bo_2|bo_2$
$b\overline{o_2}|b$
$[\![$weak sequencer$]\!]_{\mathsf{CA}}$
$[\![$weak sequencer$]\!]_{\mathsf{RA}}$

Figure 6: The *quasi weak* and the *weak sequencer* semantics.

- r is *unobtrusive* iff $[\![\rho]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r]\!]_{\mathfrak{M}}$, i.e., *if the original behaviour is preserved.*
- r is *expansive* iff $[\![\rho]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r]\!]_{\mathfrak{M}}$, i.e., *if new behaviour is added still preserving the original.*
- r is *contractive* iff $[\![\rho \bullet r]\!]_{\mathfrak{M}} \preceq [\![\rho]\!]_{\mathfrak{M}}$, i.e., *if part of the original behaviour is removed.*
- r *is* disruptive *otherwise.*

EXAMPLE. It is now possible to classify all the reconfigurations of the *sequencer* pattern mentioned above w.r.t. either the CA or RA semantic models. The $r_{proactive}$ and the $r_{dependent}$ reconfigurations are *disruptive* while the r_{weak} and the r_{Qweak} are *expansive*. To complete the examples, Figure 7 presents *unobtrusive* and *contractive* reconfigurations. Notice that the second one is classified w.r.t. the *weak sequencer* coordination pattern, and not to the *sequencer*.

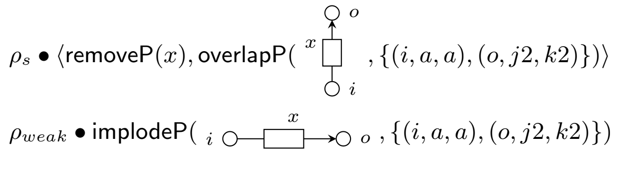

$\rho_s \bullet \langle$removeP(x), overlapP$($
o
x
i
$, \{(i, a, a), (o, j2, k2)\})\rangle$
$\rho_{weak} \bullet$ implodeP$($
i
x
o
$, \{(i, a, a), (o, j2, k2)\})$

Figure 7: Unobtrusive (top) and Contractive (bottom) reconfigurations.

SUPPORT. In practice, to compare reconfigurations from a behavioural perspective should resort to the tools available for the semantic model of interest. For CA this can be done in Vereofy [5], which will be made accessible from a reconfiguration framework through a plug-in currently under development.

4. REASONING ABOUT RECONFIGURATIONS: STRUCTURE

As discussed in the previous section, connectors exhibit behaviour, determined by the underlying semantic model, and therefore the effect of a reconfiguration can be 'measured' by whatever behavioural changes are entailed by its application. For example, in certain cases, it may be necessary to rule out reconfigurations leading to non bisimilar behaviours.

There is, however, another perspective whose focus is placed on the interconnection structure, with no reference to the emerging behaviour. Examples of *structural*, or 'syntactic' properties are, typically,

i) every FIFO_1 *channel is connected to at least a* lossy *channel* or

ii) node i is a connector's output node.

One may then require that a reconfiguration preserves such properties. This will lead to a different family of relations to compare reconfigurations. What should be remarked is the fact that such relations are independent of the underlying semantic model and, in a broader sense, not committed to the use of a specific coordination modelling language.

4.1 A Hybrid Logic

Modal logic provides the standard way of expressing properties over the graph-like structure of coordination patterns. Often, however, structural properties are to be formulated relative to a particular node in the pattern. An example is given by property *ii*) above. In general, one may require, for instance, that all the channels incident in a specific node and their interconnections remain unchanged under a reconfiguration. This justifies the choice of a (*hybrid*) logic [8] to express such properties.

Hybrid logic is a modal logic with specific mechanisms to explicitly refer to nodes in the coordination pattern. This is achieved through a set of special symbols called *nominals* through which nodes can be referred, for example to establish equalities between them or to express accessibility relationships. This is possible since each nominal is true at exactly one node in the coordination pattern. Therefore, one asserts that node n is named by (the nominal) i if i is true at n.

Besides nominals, hybrid logic introduces the @ operator, which, for a nominal i and a formula ϕ, yields a new formula $@_i\phi$ evaluating to true whenever ϕ is true in the node referred by i.

4.1.1 Syntax

Structural properties of coordination patterns will be expressed in a variant of hybrid propositional logic, where modalities are indexed by channel *types*, or, more generally, by sets of channel types. Its syntax is given by

$$\begin{aligned}\phi \ ::= \ & i \mid \mathsf{true} \mid \mathsf{false} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \\ & \langle K\rangle\phi \mid [K]\phi \mid \langle\!\langle K\rangle\!\rangle\,\phi \mid [\![K]\!]\phi \mid @_i\phi\end{aligned}$$

where $i \in \mathsf{Nom}$ and $K \subseteq \mathcal{T}$, for Nom a set of nominals. Disjunction ($\vee$) and equivalence ($\leftrightarrow$) are defined by abbreviation. For simplicity, in modalities '$-$' stands for the whole set of channel types, *i.e.*, $\mathcal{T}$, and $[-c]$, with $c \in \mathcal{T}$, stands for $\mathcal{T}\backslash\{c\}$.

Modality $\langle K\rangle$ quantifies existentially over the edges (of the graph representing the coordination pattern) labelled by channel types in K; its dual $[K] \stackrel{\text{def}}{=} \neg\langle K\rangle\neg$ provides a universal quantification. Modalities $\langle K\rangle$ and $[K]$ express properties of *outgoing* connections from the node in which they are evaluated, in a coordination pattern ρ. On the other hand, modalities $\langle\!\langle K\rangle\!\rangle$ and $[\![K]\!]$ express properties of incoming connections to that node. Finally, the @ operator *redirects* the formula evaluation to the context of a specific node, globally in the coordination pattern structure.

EXAMPLES. Property (i) is expressed as $[\mathsf{FIFO}_1]\langle\mathsf{lossy}\rangle\mathsf{true}$. Property $ii)$ is written $@_i[-]\mathsf{false}$. Notice the use of $[-]\mathsf{false}$ to state that the formula requires the absence of outgoing channels from the node referred by the nominal i.

The introduction of nominals to name channel ends, *i.e.,* nodes in a coordination pattern, makes possible to express *local* proprieties. The simplest example is to express equality between a node referred by i and another one referred by j with $@_i j$.

More complex examples encompass the expression of the property where all outgoing channels from a particular node referred by i are lossy. Formally,

$$@_i(\langle -\rangle\mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false})$$

As another example, consider formula

$$i \rightarrow \neg\langle\mathsf{sync}\rangle\langle\mathsf{lossy}\rangle i$$

which states the absence of a loop formed by a sync followed by a lossy channel at node referred by i. Notice that the absence of loops, and in general, irreflexivity of a binary relation is not expressible in classical modal logic.

Finally, the formula below requires that all output nodes are accessible through a sync channel but never through a FIFO channel:

$$[-]\mathsf{false} \rightarrow (\langle\!\langle\mathsf{sync}\rangle\!\rangle\,\mathsf{true} \wedge [\![\mathsf{FIFO}]\!]\mathsf{false})$$

4.1.2 Semantics

A model, $\mathcal{M}$, for this language is a pair $\langle\rho,\sigma\rangle$, where $\rho = \langle I, O, R\rangle \in \mathcal{P}$, and $\sigma : \mathsf{Nom} \longrightarrow \mathcal{N}$ is a naming scheme, *i.e.,* a function which assigns each nominal to a node in the coordination pattern. The satisfaction relation, of a formula by a model $\mathcal{M}$ in a node n is defined below. Notice, in particular, how nominals and the nominal satisfaction operator are handled.

$$\begin{array}{lll}
\mathcal{M}, n \models \mathsf{true} & & \\
\mathcal{M}, n \not\models \mathsf{false} & & \\
\mathcal{M}, n \models \neg\phi & \text{iff} & \mathcal{M}, n \not\models \phi \\
\mathcal{M}, n \models \phi_1 \wedge \phi_2 & \text{iff} & \mathcal{M}, n \models \phi_1 \text{ and } \mathcal{M}, n \models \phi_2 \\
\mathcal{M}, n \models \phi_1 \rightarrow \phi_2 & \text{iff} & \mathcal{M}, n \not\models \phi_1 \text{ or } \mathcal{M}, n \models \phi_2 \\
\mathcal{M}, n \models i & \text{iff} & \sigma_{\mathcal{M}}(i) = n \\
\mathcal{M}, n \models @_i\phi & \text{iff} & \mathcal{M}, \sigma_{\mathcal{M}}(i) \models \phi \\
\mathcal{M}, n \models \langle K\rangle\phi & \text{iff} & \exists_{m \in \{p \mid \langle n,-,c,p\rangle \,\in\, \rho.R \,\wedge\, c \in K\}} \cdot \\
& & \mathcal{M}, m \models \phi \\
\mathcal{M}, n \models [K]\phi & \text{iff} & \forall_{m \in \{p \mid \langle n,-,c,p\rangle \,\in\, \rho.R \,\wedge\, c \in K\}} \cdot \\
& & \mathcal{M}, m \models \phi
\end{array}$$

To capture properties relative to *incoming* connections to a node, modalities, $\langle\!\langle K\rangle\!\rangle$, $[\![K]\!]$, based on the converse of relation R of ρ are considered. Formally,

$$\begin{array}{lll}
\mathcal{M}, n \models \langle\!\langle K\rangle\!\rangle\,\phi & \text{iff} & \exists_{m \in \{p \mid \langle p,-,c,n\rangle \,\in\, \rho.R \,\wedge\, c \in K\}} \cdot \\
& & \mathcal{M}, m \models \phi \\
\mathcal{M}, n \models [\![K]\!]\phi & \text{iff} & \forall_{m \in \{p \mid \langle p,-,c,n\rangle \,\in\, \rho.R \,\wedge\, c \in K\}} \cdot \\
& & \mathcal{M}, m \models \phi
\end{array}$$

The satisfaction relation $\models$ lifts, as usual, to a notion of satisfiability by quantifying over all the nodes in the coordination pattern. I.e., ϕ is *globally satisfied* in $\mathcal{M}$ ($\mathcal{M} \models \phi$) if it is satisfied at all nodes in $\langle\rho, \mathsf{node}\rangle$.

4.2 Reconfiguration comparison

Equipped with a language to express structural properties of coordination patterns, we can define a new criterium for comparing reconfigurations. We start by defining a notion of invariance:

DEFINITION 5. *A structural property ϕ is* invariant *for a reconfiguration r iff it is preserved by r, with $\rho \in \mathcal{P}$. Formally,*

$$\langle\rho,\sigma\rangle \models \phi \;\Rightarrow\; \langle\rho \bullet r, \sigma\rangle \models \phi$$

Then,

DEFINITION 6. *Given a model $\mathcal{M} = \langle\rho,\sigma\rangle$, with $\rho \in \mathcal{P}$, and σ a naming scheme for nodes, reconfigurations r, r_1 and r_2 and a set of formulas Φ, one says*

1. *reconfiguration r* preserves *Φ iff every $\phi \in \Phi$ is* invariant *for r in model $\mathcal{M}$;*
2. *reconfigurations r_1 and r_2 are structurally equivalent with respect to Φ, written $r_1 \equiv_\Phi r_2$, iff*

$$\langle\rho \bullet r_1, \sigma\rangle \models \phi \;\Leftrightarrow\; \langle\rho \bullet r_2, \sigma\rangle \models \phi$$

for every $\phi \in \Phi$.

In practice, however, after a reconfiguration most structural relationships one may want to preserve, are *displaced*, in the sense that they remain valid but at a different node. A typical situation is illustrated in the following example.

EXAMPLE. Consider the coordination pattern whose evolution is depicted in Figure 8.a. Assume, for convenience, that nodes are referred to by their own identifiers. Clearly, at node c it is true that following a connection through a sync channel, all connections are established by lossy channels, ie

$$@_c\,\langle\mathsf{sync}\rangle(\langle -\rangle\mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false}).$$

Consider now that an insertP reconfiguration pattern is applied at node c. In a first step, node c is split because of the application of the split elementary reconfiguration (Figure 8.b). Then a new structure is linked to the nodes resulting from this operation (Figure 8.c). In both steps, however, the property is still valid for nodes cd and m_o, respectively. With respect to the example in Figure 8.c), this is expressed as

$$@_{m_o}\,\langle\mathsf{sync}\rangle(\langle -\rangle\mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false}).$$

This example motivates the following generalisation of definition 5.

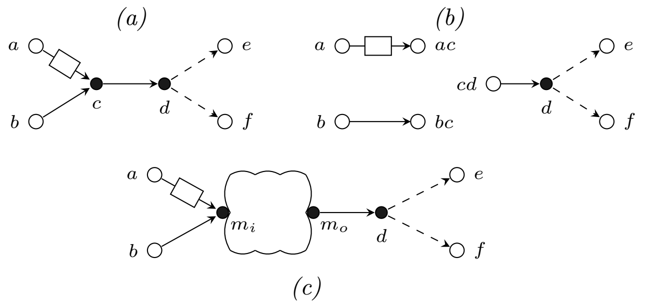


Figure 8: Example of a *displaced* invariant.

Definition 7. *Let τ be a surjection on nominals. A structural property ϕ is* invariant *for a reconfiguration r, up to a name translation τ iff*

$$\langle \rho, \sigma \rangle \models \phi \;\Rightarrow\; \langle \rho \bullet r, \sigma \rangle \models \phi[\tau]$$

where $\phi[\tau]$ stands for ϕ with all occurrences of a nominal i replaced by $\tau(i)$.

Back to the example, if $\langle \rho, \sigma \rangle \models \psi$, then $\langle \rho \bullet r, \sigma \rangle \models \psi[c \mapsto m_o]$ for r the relevant insertP reconfiguration. Notation $[c \mapsto m_o]$ denotes a function which maps c to m_o and behaves as the identity in all other cases. Naturally, equivalence $\equiv_\Phi$ can be tailored to this more general notion of invariant.

Support. Properties expressed in the hybrid logic described in this section can be verified, after a translation to classical propositional hybrid logic in the HyloRes [4] system. The translation involves a restriction of propositional symbols to nominals and an encoding of the backward modalities $\langle K \rangle$ and $[\![K]\!]$.

5. RELATED WORK

This work is in debt to previous research on the reconfiguration of Reo connectors, in particular references [10, 11]. The latter introduces a basic modal logic to reason about the constraint automata representation of Reo connectors. The former provides an axiomatisation of connector constructions, which is similar to our reconfiguration operations, to discuss connector equivalence. What distinguishes our approach is the separation between behavioural and structural concerns in a way which is, as much as possible, independent of the underlying semantic models.

Work reported in [22] resorts to category theory to model software architectures as labelled graphs of components and connectors. Reconfigurations are expressed through algebraic graph rewriting rules. In [14, 15], a similar approach is adopted, but in the context of Reo. The authors relay on high-level replacement systems, more precisely on typed hypergraphs, to describe Reo connectors (and architectures, in general). In this perspective, vertices are the nodes and (typed hyper-) edges are communication channels and components (which is quite similar to our approach). Reconfiguration rules are specified as graph productions for pattern matching. This approach performs atomic complex reconfigurations, rather than a sequence of basic modifications, which is stated as an advantage for maintaining system consistency. Nevertheless, the model may become too complex even when a simple primitive operation needs to be applied. Also in [15] Reo is encoded into mCRL2 for verification of properties of connectors expressed in the modal μ-calculus extended with data-dependent processes and regular formulas. No results are known concerning comparison and classification of reconfigurations in these approaches.

In a different setting, the DISCO framework for classification and selection of software connectors' [16] meets similar objectives in what concerns connectors key characteristics and users' needs expressed as data distributions.

6. CONCLUSIONS

The paper introduces a framework for reasoning about reconfiguration of coordination patterns. Such patterns are described as graphs of primitive channels and their reconfigurations defined through *composition* of a set of elementary operations: join, split, par and remove. Reconfigurations are assessed and compared according to two orthogonal perspectives: *behavioural*, which resorts to evaluating the reconfigured pattern with respect to the underlying semantic model (this paper resorted to Reo semantic models for illustration purposes), and *structutral*, in which properties of the interconnection graph are formulated in a variant of hybrid propositional logic. This approach provides the basis to a systematic classification of reconfiguration strategies for service-based applications, which are expected to lead to effective derivation of semi-automatic monitoring tools in the near future.

A lot of work remains, however, to be done. Providing tool support for the two reasoning perspectives is in order as well as classifying and organising reconfigurations in a suitable ontology. Moreover, it makes sense to continue working on the hybrid logic presented here by extending it to deal with properties of paths within a coordination pattern and incorporating quantitative measures to express QoS constraints. We are currently working on this topic taking into account, on a reconfiguration, suitable measures of QoS levels attached to coordination patterns [2] and to their deployment context. Relevant related work includes research on Stochastic Reo [18] and on a generic QoS algebra [7, 19, 17].

7. REFERENCES

[1] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical. Structures in Comp. Sci.*, 14(3):329–366, June 2004.

[2] F. Arbab, T. Chothia, R. van der Mei, S. Meng, Y. Moon, and C. Verhoef. From coordination to stochastic models of QoS. In J. Field and V. Vasconcelos, editors, *Coordination Models and Languages*, volume 5521 of *Lecture Notes in Computer Science*, chapter 14, pages 268–287. Springer Berlin/Heidelberg, Berlin, Heidelberg, 2009.

[3] F. Arbab and F. Mavaddat. Coordination through channel composition. In F. Arbab and C. Talcott, editors, *Coordination Models and Languages*, chapter 6, pages 275–297. Springer Lect. Notes Comp. Sci. (2315), Berlin, Heidelberg, Mar. 2002.

[4] C. Areces and J. Heguiabehere. Hylores: A hybrid logic prover based on direct resolution. In *Proc. 8th Int. Conf. on Automated Deduction (CADE-18)*. Springer Lect Notes Comp Sci. (2392), 2002.

[5] C. Baier, T. Blechmann, J. Klein, and S. Klüppelholz. A uniform framework for modeling and verifying components and connectors. In *Proc. 11th Int. Conf*

on Coordination Models and Languages, pages 247–267. Springer Lect Notes Comp Sci. (5521), 2009.

[6] C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.

[7] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[8] P. Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of IGPL*, 8(3):339–365, 2000.

[9] M. Bonsangue, D. Clarke, and A. Silva. Automata for Context-Dependent connectors. In *Proceedings of the 11th International Conference on Coordination Models and Languages*, COORDINATION '09, pages 184–203, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] D. Clarke. Reasoning about connector reconfiguration I: Equivalence of constructions. Technical report, CWI-Centrum voor Wiskunde en Informatique, Amsterdam, Feb. 2005.

[11] D. Clarke. A basic logic for reasoning about connector reconfiguration. *Fundam. Inf.*, 82:361–390, Feb. 2008.

[12] H. Gomaa and M. Hussein. Software reconfiguration patterns for dynamic evolution of software architectures. In *4th IEEE/IFIP Conf. on Software Architecture (WICSA 2004)*, pages 79–88. IEEE, 2004.

[13] P. Hnětynka and F. Plášil. Dynamic reconfiguration and access to services in hierarchical component models Component-Based software engineering. In I. Gorton, G. T. Heineman, I. Crnković, H. W. Schmidt, J. A. Stafford, C. Szyperski, and K. Wallnau, editors, *Component-Based Software Engineering*, chapter 27, pages 352–359. Springer Lect. Notes in Comp Sci. (4063), 2006.

[14] C. Krause. *Reconfigurable Component Connectors.* PhD thesis, Leiden University, Amsterdam, The Netherlands, 2011.

[15] C. Krause, Z. Maraikar, A. Lazovik, and F. Arbab. Modeling dynamic reconfigurations in Reo using high-level replacement systems. *Science of Computer Programming*, 76(1):23–36, 2011.

[16] C. A. Mattmann, D. Woollard, Nenad, and R. Mahjourian. Software connector classification and selection for Data-Intensive systems. In *Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS '07)*, page 4. IEEE, May 2007.

[17] S. Meng and L. S. Barbosa. Towards the introduction of qos information in a component model. In S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, and C.-C. Hung, editors, *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, pages 2045–2046. ACM, 2010.

[18] Y.-J. Moon, A. Silva, C. Krause, and F. Arbab. A compositional semantics for stochastic reo connectors. In *Proceedings Ninth International Workshop on the Foundations of Coordination Languages and Software Architectures*, volume 30 of *EPTCS*, pages 93–107, 2010.

[19] d. Nicola, G. L. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A process calculus for QoS-aware applications. In J. M. Jacquet and G. P. Picco, editors, *Proc. of the 7th International Conference on Coordination Models and Languages, 7th International Conference (COORDINATION 2005), Namur, Belgium, April 20-23, 2005*, volume 3454 of *LNCS*, pages 33–48. Springer-Verlag, 2005.

[20] N. Oliviera and L. S. Barbosa. Reconfiguration mechanisms for service coordination. In *Pre-Proceedings of the 9th international workshop on Web Services and Formal Methods*, pages 96–112, 2012. To appear in a Springer LNCS volume.

[21] A. J. Ramirez and B. H. C. Cheng. Design patterns for developing dynamically adaptive systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 49–58, New York, NY, USA, 2010. ACM.

[22] M. Wermelinger and J. L. Fiadeiro. Algebraic software architecture reconfiguration. In *Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-7, pages 393–409, London, UK, 1999. Springer-Verlag.

APPENDIX

A. CONSTRAINT AND REO AUTOMATA

This appendix recalls the definitions of *constraint* automata and Reo automata for the reader's convenience.

Constraint Automaton. A constraint automaton $\mathcal{A}$ is a tuple $(Q, Names, \longrightarrow, Q_0)$, where Q is a set of states, $Q_0 \subseteq Q$, is the set of initial states of $\mathcal{A}$, $Names$ is a (finite) set of names (of boundary nodes of channels or connectors), $\longrightarrow \subseteq Q \times 2^{Names} \times DC \times Q$, is the transition relation of $\mathcal{A}$, with DC being a data constraint defined by the grammar $g ::= \texttt{true} \mid d_A = d \mid g_1 \vee\ g_2 \mid \neg g$

Reo Automaton. A Reo automaton $\mathcal{A}_{\mathsf{Reo}}$ is a tuple (Σ, Q, δ), where Σ is the set of ports of a Reo connector, Q is the set of states and $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ obeys the reactivity and uniformity properties. Intuitively, reactivity means that data flows through ports with pending requests, and uniformity means that the firing set of ports is smaller or equal to the request set. $\mathcal{B}_\Sigma$ is the Boolean Algebra over Σ defined by:

$$g ::= \sigma \in \Sigma \mid \top \mid \bot \mid g \vee g \mid g \wedge g \mid \overline{g}$$