Architectural Specification of Location-aware Systems in Rewriting Logic

Nasreddine Aoumeur* José Luiz Fiadeiro

Department of Computer Science
University of Leicester
LE1 7RH, UK
{na80,jwf4}@le.ac.uk

Abstract

The concept of Location Law was recently put forward for the modelling of architectural aspects of distributed and mobile systems that need to be location-aware. These are systems for which communication and access to resources need to be modelled explicitly at the higher architectural levels of design and cannot be relegated to the lower infrastructural levels: components need to know where the components with which they interact reside and execute their computations, and how they themselves move across the distribution network. For instance, financial systems need to be location-aware because banking at a branch, at an ATM or through the internet are not just operationally different: they are subject to different business rules.

In this paper, we endow the architectural approach to location-aware systems based on Location Laws with a formal semantics expressed in terms of Meseguer's Rewriting Logic. As a result, we obtain a formal framework supporting a three-step development method that operates over two complementary architectural dimensions: the traditional coordination aspects that relate to the logical interconnection of system components, and the new location aspects that handle logical distribution and mobility. Rewriting Logic allows us to validate these dimensions separately, then against each other, and finally to superpose them on the run-time configuration of the system.

1 Introduction

The ever-increasing complexity of software-intensive systems, coupled with the need for a high degree of agility to operate in very volatile and dynamic environments, have made the construction and evolution of their high-level business *architectures* top priorities of any successful software development method. A set of architectural modelling primitives has been proposed in [AF02, AF03] for taming the complexity of construction and evolution of such systems. This approach is based on the key separation between coordination and computational aspects of systems, which allows for the more volatile aspects of the application domain to be captured as architectural connectors, so-called 'Coordination Contracts". These contracts model business rules that can be superposed, at run time, over the components that implement the core services required by the domain, allowing systems to adapt dynamically to

*Supported by the European Commission through the contract IST-2001-32747 (AGILE: Architectures for Mobility)

changes in the business context in which they operate. A software development environment has been developed for supporting the approach [AGKF02, GKW+02]. Its use in industrial-size projects has also been documented [WKL+04, WKA+03].

However, the increase on the popularity of mobile devices (phones, PDAs) and the proliferation of new channels for doing business [MPS03] have added a new dimension of complexity to software development. Many systems today need to be location-aware in the sense that communication and access to resources need to be modelled explicitly at the higher architectural levels and cannot be relegated to the lower infrastructural levels: components need to know where the components with which they interact reside and execute their computations, and how they themselves move across the distribution network. For instance, financial systems need to be location-aware because banking at a branch, at an ATM or through the internet are not just operationally different: they are subject to different business rules. In other words, there is a logical, business-level notion of location that needs to be reflected in the higher architectural levels of systems.

To cope with this new distribution/mobility dimension, new architectural primitives have been investigated in the IST-FET-GC1 project AGILE [ABB+03]. These primitives were first characterised at the level of the prototype language CommUnity [FL04, LFW02]. Their semantics was defined in [LF03] based on the categorical semantics of CommUnity [FLW03, Fia04]. More recently, these developments were used for extending the coordination-based architectural approach of [AF02, AF03] with so-called Location Laws and Contracts. These new modelling primitives for location-awareness have been motivated and characterised at an intuitive level in [AFO04b]. More specifically, we demonstrated how a coordination/location-driven architectural approach can be put to an effective use in modelling distributed business processes within the service-oriented computing paradigm [AFO04a].

Our purpose in this paper is to endow this coordination/location-driven architectural approach with a formal underpinning that complements the (declarative) semantics developed over CommUnity [LF03] in order to cater for the more operational aspects of the underlying development process, including validation and rapid-prototyping. The semantic framework that we propose is expressed in terms of Meseguer's Rewriting Logic [Mes92, Mes98].

At this first stage of consolidation of the approach, the benefits of opting for Rewriting Logic are multiple:

1. It provides a general framework where many models of true concurrency have been axiomatized [MOM96]. The architectural modelling techniques of CommUnity have also been formalized in Rewriting Logic [FMOM+00];

2. It is an "operational" logic allowing rapid prototyping through concurrent rewriting techniques [DJ90]. These capabilities have been pushed further through the development of the specification/programming Maude language [CDE+99];

3. It is reflective by essence [CM96]. These reflection capabilities may be exploited for controlling rewriting executions through strategies [BKKM02], for dynamically evolving specifications as shown in the Co-nets framework [AS02, AS04], and for modelling mobility as demonstrated with mobile Maude [DELM00].

We start by proposing a stepwise methodology for governing the approach based on three steps: (1) Coordination and location laws start by being interpreted independently and

validated against possible misunderstandings or errors using an implementation of the MAUDE language; (2) Both concerns are then integrated and interpreted into an adequate rewrite theory, after which they are validated against possible inconsistencies and interference; (3) Finally the integrated and validated concerns are superposed over computational components using suitable inference rules.

The rest of this paper is organized as follows. First, a short overview of the underlying architectural approach is given. In the third section, more motivation about Rewriting Logic and the stepwise development methodology is presented. Then, a complete section is devoted to each development step. Finally, we situate our work with respect to related approaches, summarise what we have achieved and highlight the steps that have been planned for its extension and enrichment.

2 The Coordination/Location-driven Approach

Using a very simplified part of a banking system example, we start by recalling the essence of the coordination-based modelling primitives. In the second part we present the location-based conceptual primitives and the way both concerns have to be integrated to model the complete behavior of systems.

2.1 Coordination concerns : An overview

Coordination primitives have been put forward [AF02, AF03] for separating interactions from computations. More precisely, so-called coordination laws and contracts externalize as first-class entities any (intra- or cross-organisational) interactions between involved components. With respect to information systems, this separation occurs between the computations performed by relatively stable core entities and the interconnections that model the more volatile business rules. This clean separation permits changes to the business rules to be localized only on the volatile parts, with a minimal impact on the core services.

The way coordination aspects can be captured as architectural connectors has been reported in several publications, e.g. [AF02]. From a conceptual modelling point of view, they are captured in semantic primitives called Coordination Laws.

Coordination Law : A coordination law captures the way a business rule requires given classes of business entities (partners) to interact; the partners are not identified at the instance level: the law identifies the roles played by these partners through generic "coordination interfaces". The way interactions between partners are coordinated according to the business rule is captured in the form of event-condition-action (ECA) rules [KL04, WKL03]; auxiliary attributes and operations may be defined when needed.

Coordination Interface : A coordination interface identifies what is normally called a role of a connector; it consists of the set of required services, events and invariants that have to be (directly or indirectly through refinement/renaming) provided by a corresponding business entity to become coordinated as described by the interaction rules of the law.

Example 2.1 For illustration, we follow a simplified application dealing with activities related to banking (e.g. deposit, withdraw, transfer, etc). For the withdrawal activity, for

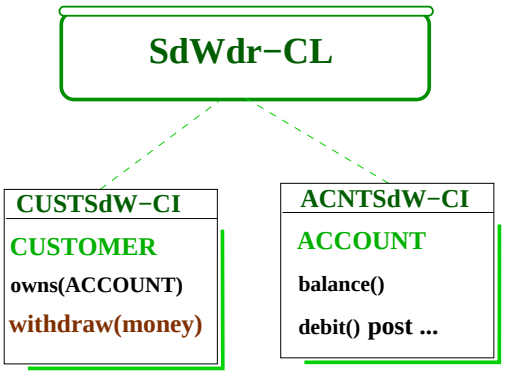
instance, we may consider different Coordination Laws, each of which captures a particular business rule relating a customer with an account. For instance, one may distinguish a "standard"-withdrawal in which the customer can only request amounts less than the current balance, or a "VIP"-withdrawal where a credit limit can be used when withdrawing amounts greater than the balance.

In what follows, we elaborate on these two business rules for withdrawing money. In the case of the standard-withdrawal, the corresponding business rule can be captured as a coordination law in the following way:

```
coordination interface CUSTSdWW-CI
partner type CUSTOMER
services   owns(a:ACCOUNT):Boolean
events   withdraw(n:money; a:ACCOUNT)
end interface
```

```
coordination interface ACNTSdW-CI
partner type ACCOUNT
services
  balance():money
  debit(a:money)
     post balance() = old balance()-a
end interface
```

```
coordination law SdWdr-CL
partners acco:ACNTSdW-CI; cust:CUSTSdW-CI
rule Withdraw
when cust.withdraw(n,acco)
      with acco.balance() ≥ n &
            cust.owns(acco)
      do acco.debit(n)
end law
```



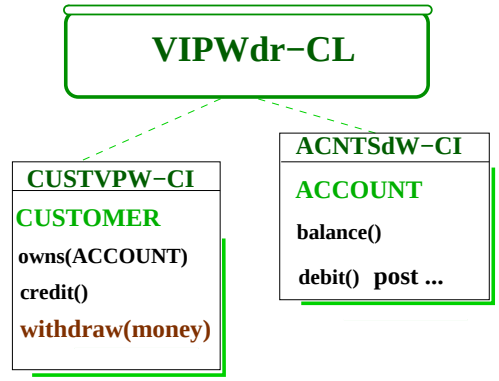
The coordination law for standard withdrawals requires, as expected, two coordination interfaces providing for the account and customer business entities. For this specific withdrawal, the customer interface denoted by `CUSTSdW-CI` requires that the customer component provides the testing of `owner`ship of a given account as a service, and the publication of the event `withdraw`. The account interface denoted by `ACNTSdW-CI` requires that the account component provides as services the `balance` as well as an operation for `debit`ing. Requirements on this latter service are specified in terms of pre- and post-conditions.

The coordination law itself first states that the partners that it can coordinate are the instances two interfaces (i.e. `CUSTSdW-CI` and `ACNTSdW-CI`). Then, through ECA rules, it specifies that once receiving the withdraw event, the ownership and availability of money are first checked, in which case the requested amount is debited. Otherwise, the request is refused. ♢

This very elementary coordination law can be used to illustrate some of the aspects of our approach to dynamic evolution. Instead of incorporating the conditions on which an account may be debited by its owners in the core business entity account, they are externalized in the coordination law so that they can *evolve* independently to capture different forms of withdrawal. This means that the core business entities remain very basic, so that interaction always prevails upon internal hard-coded computation; for instance the debit here consists just in decrementing the balance by the requested amount.

Example 2.2 For instance, in order to offer a VIP-withdrawal in which a given credit limit is allowed, we just have to change the interaction as modelled by the coordination rule; the basic debit service does not need to be changed.

```
coordination law VIPW-CL
partners acco:ACNTSdW-CI; cust:CUSTVPW-CI
rule :   Withdraw
when cust.withdraw(n,acco)
      with acco.balance()+cust.credit()≥n
          and cust.owns(acco)
      do  if acco.balance() ≥ n
          then acco.debit(n)
          else acco.debit(1.01n)
end law
```



It is worth mentioning that the coordination interface that captures the role of the account remains unchanged, which emphasises the fact that the change of business rule does not imply changes on the core account entity. However, a different interface captures the role of the customer: we need an additional service that returns the credit limit currently assigned to the customer, meaning that customers who have not been provided with a credit limit cannot be coordinated by this law. This coordination interface is defined as follows:

```
coordination interface CUSTVPW-CI
partner type CUSTOMER
services
      owns(a:ACCOUNT):Boolean
      credit():money
events   withdraw(n:money; a:ACCOUNT)
end interface          ♢
```

2.2 Location Concerns

As emphasized in the introduction, the purpose of location primitives is to enhance architectural mechanisms to deal not only with interactions but also with an orthogonal distribution/mobility dimension. As a result, we can model and reason about different business channels, communication infrastructure and other advanced ICT-based constituents (e.g. mobile devices, sensors, etc). That is, just like with the coordination mechanisms, we want to define location primitives that can externalise any added-value and/or constraints related to the distribution topology over which services are composed.

In this respect, the properties that are determinant for an explicit characterization of the distribution dimension include in particular:

- The communication status, i.e the presence, absence, or quality of the communication link between locations where given services are executing and require data to be exchanged or need to be syncronized as part of the distribution strategy.

- The ability to continue the execution of an activity at another location, which requires that the new location is reachable from the present one so that the execution context can be moved.

By capitalising on the work developed around CommUnity [LF03], these two characteristics are captured through two primitives:

- Communication is captured through the "be-in-touch" construct $BT : set(LOC) \times LOC \rightarrow BOOL$, a boolean operation over any two (or more) involved locations. LOC is a sort through which the types of location that are relevant for the application at hand can be specified.
- Reachability is captured by the operation $REACH : LOC \times LOC \rightarrow BOOL$ that returns whether a given location is reachable or not from another one.

Location laws are the means through which we model distribution concerns in business architectures. Like in coordination laws, we use ECA-rules for capturing the dependency of the behaviour of the system on the properties of the distribution topology. Location interfaces capture the features required of the locations involved in a given business rule.

As just emphasized, neither the presence nor the quality of communication can be taken for granted in the distribution dimension. Depending on availability of communication and reachability, either a full composition involving all features (i.e. services, triggers, etc) from participating partners can be achieved, or just a composition of features at the current location of the trigger is possible.

Example 2.3 Let us consider the withdrawal activity from the distribution perspective. The location where the withdrawal is requested is crucial for determining how the system reacts. That is, the properties of the business channel (e.g. ATM, internet, branch, etc) and the status of the observables (`BT` and `REACH`) become relevant. What is important for the distribution dimension is not that the customer has a standard or a VIP-contract with the account but, for instance, whether the ATM at which the request for the withdrawal is made can communicate with the branch in which the account is held. In this case, the distribution law will determine how much money can be given according to the context in which the transaction is being made (cash available at the ATM and status of the communication between the ATM and the branch).

For that purpose, the services that we require from the ATM location partner consist of: (1) the amount of cash available inside the machine; (2) the default maximum amount that the machine gives if there is no connection to the account; and (3) the service that gives the requested money. Also, (4) the account number available to the ATM after inserting the card (i.e. at the identification activity [AFO04a]) is required. Along with the event for requesting the withdrawal, these services constitute the ATM location interface that we denote by `ATMWdr1-LI`.

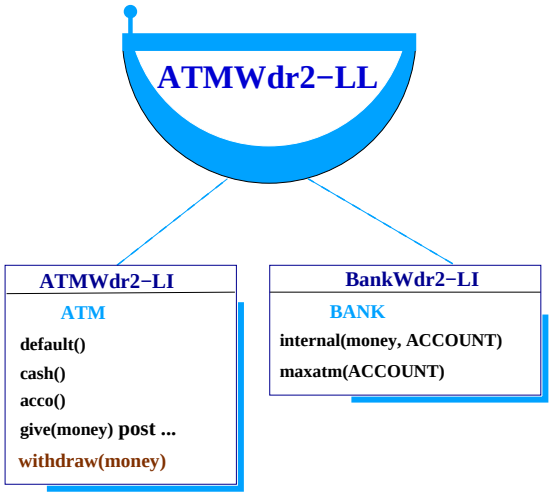
The location interface that applies to the bank is as below. That is, the bank is required to make available, for every account, the maximum amount that can be debited from an ATM, as well as a service that captures the internal operation or the code of debiting an amount of money (from an account).

```
location interface ATMWdr2-LI
location type ATM
services
      default(), cash():money
      acco():ACCOUNT
      give(n:money)
        post cash() = old cash()-n
events    withdraw(n:money)
end interface
```

```
location interface BANKWrd2-LI
location type BANK
services
      internal(n:money, a:ACCOUNT)
      maxatm(a:ACCOUNT): money
end interface
```

These two location interfaces are brought together in the location law that defines the distribution concerns of the withdrawal activity when performed at an ATM with the above capabilities. In this law, the fact that the `atm` and the `bank` locations are not "in touch" (`BT`) does not mean that one cannot be reached from the other (`REACH`). Reachability allows for mobility of services, namely for service execution to be moved to other locations as an instance of another service. In the case that concerns us, even in the absence of communication with the bank, ATMs can provide a limited amount of cash as long as there is a protocol with the bank for remote/delayed transmission of the corresponding withdrawal. The operations that continue the execution of the activity at a different location are declared under `mv` whereas those that are executed locally are identified under `do` as usual.

```
location law ATMWdr2-LL
locations bank:  BANKWdr2-LI; atm:  ATMWdr2-LI
rule :   Withdraw
when atm.withdraw(n)
   and BT(atm,bank)
   with  n ≤ bank.maxatm(atm.acco())
         and n ≤ atm.cash()
   do atm.give(n)
when atm.withdraw(n) and
   ¬ BT(atm,bank)
   and REACH(atm, bank)
   let N=min(atm.default(),n) in
   with N≤atm.cash()
   do atm.give(N)
   mv bank.internal(N, atm.acco()))
end law◇
```



2.3 Integration of concerns

In the two the previous subsections, we discussed a set of semantic primitives through which we can separate two different concerns in modelling/evolving behavioural features of software systems:

- The coordination mechanisms that need to be put in place to establish the interactions involving services from different partners (e.g. software components, business entities, etc);
- The location mechanisms that reflect the added-value and constraints brought and/or imposed by the distribution infrastructure in place in terms of locations (e.g. mobile devices, PDA, business channels) from which the partners have to interact.

Because both coordination and location laws are specified using ECAs, that is, in an event-driven way, it is straightforward to *unify* all the laws triggered by semantically identical events in order to reflect the complete behavior of any activity.

This unification is expressed in terms of a synchronisation of all location and coordination laws around the similar triggering event (and similar activity).

Example 2.4 For our withdrawal example, the event that triggers the withdrawal business activity instantiates as `atm.withdraw(n)` in the location interface and `cust.withdraw(n,acco)` in the coordination interface. Assuming that the coordination law that is active in the run-time configuration is SdWdr-CL (i.e. standard withdrawal), the occurrence of the event is subject to the following rules:

```
when cust.withdraw(n,acco)
     with acco.balance() ≥ n
         and cust.owns(acco)
     do acco.debit(n)
```

```
when atm.withdraw(n) and BT(atm,bank)
     with  n ≤ bank.maxatm(atm.acco())
         and n ≤ atm.cash()
     do atm.give(n)
```

```
when atm.withdraw(n) and ¬ BT(atm,bank) and
     REACH(atm,bank)
     let N=min(atm.default(),n) in
     with N=atm.cash()
     do atm.give(N)
     mv bank.internal(N, atm.acco()))
```

The joint execution of ECA rules that we have in mind, as formalised in [FLW03], takes the conjunction of the guards and the parallel composition of the actions (i.e. the union of the corresponding synchronisation sets) when BT holds—as shown in the left-hand side below. When the located partners are not in touch, i.e. cannot communicate, the coordination rules do not apply—as shown in the right-hand side below. As a result, the rules according to which a withdrawal is performed are:

```
when atm.withdraw(n) and BT(atm,bank)
     with  n ≤ bank.maxatm(atm.acco()) and
       n ≤ atm.cash() and
       n ≤ acco.balance() and cust.owns(acco)
     do atm.give(n) and acco.debit(n)
```

```
when atm.withdraw(n) and
     ¬ BT(atm,bank) and
     REACH(atm,bank)
     let N=min(atm.default(),n) in
     with N=atm.cash()
     do atm.give(N)
     mv bank.internal(N, atm.acco()))
```

That is, when the ATM is in communication with the bank, the withdrawal is performed according to the coordination rule of a standard withdrawal and the location rule of the ATM. Notice, however, that `cust.owns(acco)` holds as a result of the binding and, hence, was omitted from the with condition. The need for communication is obvious in the guard condition, which requires the balance of the account to be checked, and the action, which requires the account to be debited. In the case of the joint execution of the guard, `BT` is necessary to ensure synchronous, atomic execution of the reaction. Notice that synchronous execution does not involve `REACH` because the service is not being moved from one location to another: both services are executed, each in its location, but atomically, which is what requires communication. ♢

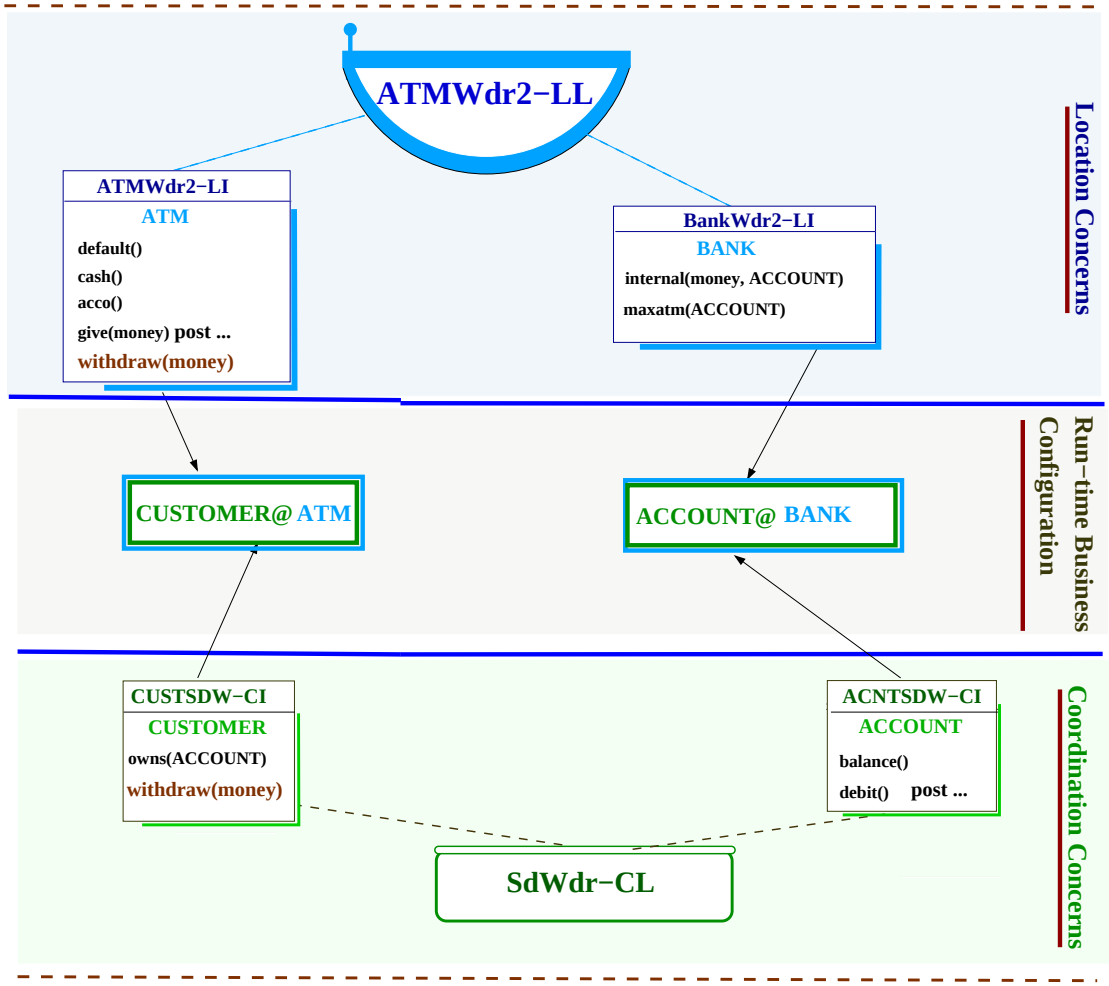


Figure 1: The Architecture illustrated with the withdrawal activity.

3 The C/L approach in Rewriting Logic: Methodology

This section is devoted to the theoretical underpinnings of the Coordination/Location-driven architectural approach (shortly, the C/L approach) that we have outlined in the previous section. Our objective is to propose a semantic framework that meets the following objectives:

1. To interpret and check both coordination and location laws independently for inconsistencies, errors, and so on. This interpretation and validation should take place intra-coordination/location (i.e. coping with one law at a time) and inter-coordination/location (i.e. validating different coordination/location laws against each other to detect inconsistencies and interference).

2. After achieving such separate semantic interpretation and validation, to validate their integration.

3. To superpose these integrated and validated coordination/location concerns on computational components of the run-time configuration of the system.

The semantic framework we are proposing to fulfil these objectives is based on Rewriting Logic [Mes92]. This is a formalism that combines declarative and operational semantics

of concurrent/distributed computation [MOM96, Mes98]. The MAUDE language [CDE+99] allows such computations to be programmed as theories in this logic using inference to capture the operational semantics.

More precisely, accordingly to the above set of objectives, we propose a methodology consisting of four incremental steps:

- First, we show how any coordination or location law can be easily expressed in Rewriting Logic using MAUDE, and thus be validated using the current implementation of this language. This validation concerns the internal functioning of any coordination or location law, and allows for detecting misunderstandings and errors while conceiving these laws by using the associated sequent of proofs [MT99, Den98].

- To allow reasoning about more than one (coordination/location) law at the same time, thus validating them against inconsistencies and interference, the next step is to derive compositional rewrite theories from those introduced in the first step.

- After being able to reason about and validate coordination and location laws independently, in this third step we propose transformational inference rules that permit to integrate both concerns following the informal guidelines presented in the previous section.

- The last step consists in putting forward inference rules to superpose the coordination and location laws on computational components of the run-time environment.

In the subsequent sections we detail each step separately, starting with an informal explanation and illustration when required. The rest of this section is devoted to recalling some features of Rewriting Logic in general and, more specifically, the MAUDE language.

3.1 Introduction to Rewriting Logic and MAUDE

Rewriting Logic was developed by J. Meseguer in [Mes90], and has become a widely accepted unified model of concurrency [Mes98]. This logic is based on two straightforward and powerful ideas: first, in contrast to equational theory, Rewriting Logic interprets each rewrite rule not as an oriented equation but rather as a change or becoming through a powerful categorical foundation [Mes92]. Second, it proposes to perform the rewriting process with maximal concurrency on the basis of four inference rules (see [Mes92] for a more detail).

In addition, one of the advantages of Rewriting Logic lies on its capability for unifying the object paradigm with concurrency without suffering from the well-known inheritance anomaly [Mes93b]. This conceptualization goes together with a very rich object-oriented programming/specification language called MAUDE [CDE+99].

In MAUDE, the structural as well as the functional aspects are specified algebraically using notations that are very similar to OBJ [GWM+92], while the dynamic aspects are captured by so-called system modules that can be elegantly denoted using object modules.

Object states in MAUDE are conceived as terms —precisely as tuples— of the form $\langle Id : C | atr_1 : val_1, .., atr_k : val_k \rangle$; where Id stands for the object identity and it is of a sort denoted by OId, C identifies the object class and is of sort CId; $atr_1, ..., atr_k$ denote the attribute identifiers of sort AId, and $val_1, ..., val_k$ denote their respective current values and are of sort $Value$. The messages (i.e. method invocation) are regarded as operations

sent or received by objects, and their generic sort is denoted Msg . Object and message instances flow together in the so-called configuration, which is no more than a multiset, w.r.t. an associative commutative operator denoted by '_ _', of messages and (a set of) objects. The simplified form of such configuration[1], described as a functional module takes the following form [Mes90]:

```
fmod Configuration is
  protecting ID       **** provides OId, CId and AId .
  sorts Configuration Object Msg .
  subsorts OId < Value .
  subsorts Attribute < Attributes .
  subsorts Object Msg < Configuration .
  op _:_ :  AId Value → Attribute .
  op _,_ :  Attribute Attributes → Attributes [associ.  commu.  Id:nil]
  op ⟨_:_|_⟩ :  OId CId Attributes → Object.
  op _ _ :  Configuration Configuration → Configuration [assoc comm id:null].
endfm.
```

The effect of messages on objects to which they are addressed is captured by appropriate rewrite rules. The general form of such rules, known as a communication event pattern, takes the following form:

$$r(\bar{x}) : M_1..M_n\langle I_1 : C_1|atts_1\rangle...\langle I_m : C_m|atts_m\rangle \longrightarrow \langle I_{i_1} : C'_{i_1}|atts'_{i_1}\rangle...\langle I_{i_k} : C'_{i_m}|atts'_{i_k}\rangle\langle J_1 : D_1|atts'_1\rangle...\langle J_{i_k} : D_p|atts'_p\rangle M'_1...Ms'_q \text{ if } C$$

where r is the label, and $\bar{x}$ is the list of the variables occurring in the rule, the Ms are message expressions, $i_1, .., i_k$ are different numbers among the original $1, .., m$, and C is the rule's condition. That is, a number of objects and messages can come together and participate in a transition in which some objects may be created, other may be destroyed, and other can change their state, and where new messages may be created.

In Maude, subclasses may be specified using the notion of class inheritance or module inheritance. Class inheritance is semantically captured by subsorts and technically materialized by the notion of attribute variables denoted by $ATTS$. Module inheritance is used for allowing (different forms of) overriding of superclass(es) methods in the definition of subclass(es).

Finally, it is worth mentioning that in Maude object instances are explicitly created (and deleted) with guaranteeing the uniqueness of their identities (modeled as natural counters) using an appropriate rewrite rule. However referring to the above described rewrite form, 'implicit' creation or deletion is also possible.

Example 3.1 A rigid object-oriented based specification of the banking main methods (e.g. withdrawal, deposit, transfer) could be obtained through the following Maude object-oriented module.

```
omod ACCNT is
  protecting REAL
```

[1] A complete description of the notion of configuration is given in [Mes93a].

```
  class Accnt | bal :  NNReal .
  msgs  deposit  withdraw:  OId  NNReal → Msg .
  msg  transfer_from_ :  NNReal  OId  OId → Msg .
  vars A  B : OId .
  Vars  M N N': NNReal .

    ********* The Account behaviour.
  rl deposit(A,M) ⟨A : Accnt|Bal : N⟩ ⇒ ⟨A : Accnt|Bal : N + M⟩ .
  rl withdraw(A,M) ⟨A : Accnt|Bal : N⟩ ⇒ ⟨A : Accnt|Bal : N − M⟩ if N ≥ M .
  rl transfer M from A to B ⟨A : Accnt|Bal : N⟩ ⟨B : Accnt|Bal : N′⟩ ⇒
          ⟨A : Accnt|Bal : N − M⟩⟨B : Accnt|Bal : N′ + M⟩ if N ≥ M .
endom
```

In this specification, first we assume that at the data level, the real data type has been specified (with `NNReal` as subsort for positive real to specify money). For instance, the rule

$$\texttt{deposit}(\texttt{A},\texttt{M})\langle A : Accnt|Bal : N\rangle \Rightarrow \langle A : Accnt|Bal : N + M\rangle$$

says that when an object state is deposited using the message $deposit(A, M)$, the next state results in increasing the balance with the deposited amount of money. We note that, due to inter-object concurrency different objects can perform these rules simultaneously when respective messages are present in the configuration. ♢

4 Rewrite theories for coordination/location laws

4.1 Coordination laws in Rewriting Logic

First we present at an informal level the conceptualization of coordination laws using the Maude language. Afterwards, we develop a rewrite theory as a specialization of the general rewrite theory fitting the characteristics of coordination laws.

4.1.1 Informal presentation

The straightforward way to capture the syntactical features of a given coordination law is to consider each law and associated interfaces as Maude classes. More precisely:

- The attributes of a given interface have to constitute an (abstract) object (state) for that interface, whereas the services are considered as messages for that class. Besides that, we distinguish between messages and events as features to be added to the Maude syntactical configuration seen above.

- The coordination contract state is constructed in the same way except that it has to include the types of the interface classes (partners) as composite attributes.

The ECA rules characterizing a given coordination law are captured as rewrite rules in the following way:

when-part : The events in the when part, modelled as (special) messages, are part of the left hand-side of the rule. We note that when a complex composition of events is at stake, we can adopt the algebras proposed in [Lec96].

with-part : First, all the attributes involved in the condition have to be grouped in the left-hand side of the rule as object states. Depending on their origin, they correspond

to interface or contract states. Second, the condition itself becomes the condition part of the rule.

do-part : The operations involved are part of the right-hand side of the rule. The states involved in the with part also have to be included in the right-hand side (with new attribute values when it is the case).

Finally, we note that pre- and post-conditions associated with services are translated into rewrite rules in the same way.

Example 4.1 Following the steps described above, the coordination law corresponding to the standard withdrawal is specified in MAUDE as follows. First, we associate the two coordination interfaces with the following MAUDE object classes.

```
coordination interface CUSTSdWW-CI
partner type CUSTOMER
services   owns(a:ACCOUNT):Boolean
           CsName:String
events   withdraw(n:money; a:ACCOUNT)
end interface
```

```
omod CUSTSdWW-CI is
  protecting money  CUSTOMER  BOOL.
  subsorts ACOID  COID < OID .
  class CUSTSdWW-CI | CsName:String .
  msgs  owns :  COID  ACOID → BOOL .
  evnts  withdraw :  COID  money  ACOID  → Evnt .
  vars Nm :  String .
endo
```

As mentioned above we are using the sort `Evnt` as a new message type to distinguish events from messages. This could slightly be achieved by updating the configuration specification recalled in the previous section. We have added the name of the customer `CsName` as a service in the interface so that the associated class has at least one attribute. Finally, we have specialized the general object identity sort `OID` to `COID` and `ACOID` to capture the specificity of customer and account identities. That is, these sorts can now be specified to capture any specific property related to these identities.

All the objects to which an event or message is addressed should now be explicitly present. For instance, for both `owns` and `withdraw` we added the parameter `COID` (i.e. customer identity) as they are addressed to the customer. Finally, variables may be declared to be used later on in the rule associated with the law.

```
coordination interface ACNTSdW-CI
partner type ACCOUNT
services
  bal():money
  debit(a:money)
     post bal() = old bal()-a
end interface
```

```
omod ACNTSdW-CI is
  protecting money ACCOUNT .
  subsorts ACOID  COID < OID .
  class ACNTSdW-CI | bal :  money .
  msgs  debit :  COID  ACOID  money → Msg .
  vars acco :  ACOId .
  vars B, m :  money .
********* Pre- and Post- condition as a rewrite rule.
    rl debit(m) ⟨acco|Bal : B⟩ ⇒ ⟨acco|Bal : B − m⟩ .
```

In this translation, we note that the post-condition associated with the debit service is captured by a rewrite rule. The law itself is translated as follows:

```
coordination law SdWdr-CL
partners acco:ACNTSdW-CI;
         cust:CUSTSdW-CI
rule Withdraw
when cust.withdraw(n,acco)
      with acco.bal() ≥ n &
            cust.owns(acco)
      do acco.debit(n)
end law
```

```
omod SdWdr-CL is
  using ACNTSdW-CI  CUSTSdW-CI.
  protecting Money .
  subsorts CTOID < OID .
  class SdWdr-CL | ACNTSdW-CI, CUSTSdW-CI .
  vars ac :  ACOId .
  vars B, m :  Money .
  vars sdwdr:  CTOID .
  vars acco:  ACNTSdW-CI.
  vars cust:  CUSTSdW-CI .
********* the coordination rule as a rewrite rule.
  rlwdr:  withdraw(cust, acco, n)
  ⟨acco|bal : B⟩⟨cust|CsName : Nm⟩⟨sdwdr|acco, cust⟩
     ⇒ ⟨acco|bal : B⟩⟨cust|CsName : Nm⟩⟨sdwdr|acco, cust⟩
     debit(cust, acco, n)  if  B ≥ n ∧ owns(cust, acco) = true  .
```

In this specification the following points need an additional explanation. First, notice that the attributes of the contract class are the two interface class sorts: we have added a declaration line for `using` them. The rule involves all the attributes from the interface and the contract states concerned. ♢

Finally, it is important to emphasize the fact that the interface classes should be instantiated only by superposing them on concrete running components. However, for the validation purposes, we can instantiate them 'artificially' with any user values (i.e. attribute values and service instances). Through this rapid-prototyping technique, we can execute the rewrite rule associated with the coordination law. check it against misunderstandings and errors at an early stage, and correct them accordingly. This is, indeed, the main objective of this first step.

4.1.2 Rewrite theory for coordination laws

We can now propose a tailored rewrite theory that allows us to reason about coordination laws in Rewriting Logic. See for instance [MT99] for some of the benefits of specializing rewrite theories. We start by capturing the above intuitive descriptions of Cor-laws-state and -template, as follows:

Definition 4.1 (Cor-laws-state signature) A coordination law state signature is defined as a pair $(S_D \cup S_C \cup S_{P_c}, \{Op\}_{S_C} \cup \{Op\}_{S_{P_c}})$ with:

- S_D is a set of (data) sorts with at least: $\{bool, COId, POId, AId, Value\} \subset S_D$. To allow object-attribute values, OId is defined as a subsort of $Value$ (i.e. $OId < Value$), with $Value$ a generic sort of values. $COId, POId$ are subsorts of OID referring to contract and partners identities sorts.
- S_C is a set of coordination sorts (different from S_D), which we assume contains at least one sort.
- S_{P_c} is a set of partners (i.e. interfaces) sorts (different from S_D and S_C). We assume S_{P_c} contains at least one sort.
- $\{Op\}_{S_C}$ is a set of 'coordination state' operations indexed by $COId \times (POId \times S_{P_c})^n \times (AId \times Value)^+ \times S_C$. More precisely, with each coordination sort from S_C a 'coordination

state' operation is associated with n partners ($n \geq 1$).

- $\{Op\}_{S_{P_c}}$ is a set of 'partner state' operations indexed by $POId \times (AId \times Value)^+ \times S_{P_c}$. More precisely, with each partner sort from S_{P_c} a 'partner state' operation is associated. AID stands for attribute identifier sorts. $\Diamond$

We should note here that both $\{Op\}_{S_C}$ and $\{Op\}_{S_{P_c}}$ are the (contract and partner) object states and are represented as tuples (as given the MAUDE illustration above). On the basis of this notion of COR-LAWS state signature, a COR-LAWS template signature is defined by extending it by (partner/contract) message and event sorts and related message operations.

Definition 4.2 (COR-LAWS-template signature) A template signature is defined as a pair $(S_D \cup S_C \cup S_P \cup S_{Msg_c} \cup S_{Msg_{p_c}} \cup S_{Evnt_{p_c}}, \{Op\}_{S_O} \cup \{Op\}_{S_{Msg_c}} \cup \{Op\}_{S_{Msg_{p_c}}} \{Op\}_{S_{Evnt_p}})$ with:

- $(S_D \cup S_O \cup S_C, \{Op\}_{S_O} \cup \{Op\}_{S_{P_c}})$ is a COR-LAWS state signature as defined above.
- $S_{Msg_c} \cup S_{Msg_{p_c}}$ are a disjoint sets of 'message generator' sorts. They correspond to the operation sorts defined in the coordination laws and interfaces respectively.
- $\{Op\}_{S_{Msg_c}}$ (resp. $\{Op\}_{S_{Msg_{p_c}}}$), is a set of message operations. That is, they are operations indexed by $COId \times S_D^* \times S_{Msg_c}$ (resp. $POId \times S_D^* \times S_{Msg_{p_c}}$).
- $\{Op\}_{S_{Evnt_{p_c}}}$, is a set of event operations. That is, they are operations indexed by $POId \times S_D^* \times S_{Evnt_{p_c}}$. $\Diamond$

Template specification. Given a COR-LAWS-template signature denoted by CTS that captures the structural aspects of coordination laws, its behaviour is constructed by associating it what we call a COR-LAWS-rule system $\mathcal{RC}$ and corresponding COR-LAWS-rewrite theory $\mathcal{RT}_{\mathcal{C}}$. —leading to the notion of COR-LAWS-template specification that we denote by $SPC =\prec CTS, \mathcal{RC} \succ_{\mathcal{RT}_{\mathcal{C}}}$. First, the following notation conventions are needed.

Notation 4.1

1. We denote by $T_{S_C}(X_{Value})$ (resp. $T_{S_{P_c}}(X_{Value})$) the set of algebraic terms[2] of sort(s) S_C (resp. S_{P_c}); where X_{Value} is an s_atc_i-(resp. s_atp_i) indexed set of variables; with s_atc_i and s_atp_i the specific sorts of attributes associated with each coordination or partner state sort. That is, for each (coordination or partner) attribute sort we assume given a set of variables. In the sequel we just use X as a union of all these sets of variables. By $T_{S_C}(\emptyset)$ (resp. $T_{P_c}(\emptyset)$) we denote their associated (ground) object terms.

2. Similarly, we will denote by $T_{Msg_C}(X)$ (resp. $T_{Msg_{P_c}}(X)$ and $T_{Evnt_{P_c}}(X)$) the set of algebraic terms of sort(s) S_{Msg_C} (resp. $S_{Msg_{P_c}}$ and $S_{Evnt_{P_c}}$).

3. To represent different multi-sets of terms (in states or rules), we denote by $MT_{S_C}(X)$ and (resp. $MT_{Msg_C}(X)$) the multiset of terms over $T_{S_C}(X)$ (resp. $T_{Msg_C}(X)$), with the 'blank' binary $\_\_$ as union (associative and commutative) operation, and $\emptyset_M$ as the identity element. The two multiset forms will be subsequently referenced by $[T_{S_C}(X)]$ and $[T_{Msg_C}(X)]$.

[2] As usual constants and variables are terms, and if $t_1, .., t_n$ are terms then $f(t_1, ..., t_n)$ is a term, with f as an n-ary operator.

4. In the same way we denote by $MT_{P_C}(X)$ and (resp. $MT_{Msg_{P_c}}(X)$ and $MT_{Evnt_{P_c}}(X)$) the multiset of terms over $T_{S_{P_c}}(X)$ (resp. $T_{Msg_{P_c}}(X)$ and $T_{Evnt_{P_c}}(X)$), with the 'blank' binary $\_\_$ as union (associative and commutative) operation, and $\emptyset_M$ as the identity element. The three multiset forms will be subsequently referenced by $[T_{P_C}(X)]$, $[T_{Msg_{P_c}}(X)]$ and $[T_{Evnt_{P_c}}(X)]$.

Definition 4.3 (Cor-laws**-states)** A Cor-laws state is any element[3] of $[T_{S_C}(\emptyset) \cup T_{S_{P_c}}(\emptyset) \cup T_{Msg_c}(\emptyset) \cup T_{Msg_{p_c}}(\emptyset) \cup T_{Evnt_{p_c}}(\emptyset)]$. $\Diamond$

Cor-laws **rewrite theory.** We are now ready to propose a Cor-laws-*rewrite theory* as a particular instantiation of the general rewrite theory given in [Mes92].

Definition 4.4 (Cor-laws **rewrite theory)** Given a Cor-laws-template-signature as presented above, a Cor-laws rewrite theory is a set of quintuples $\mathcal{RC} \subset L_{CLI} \times ([T_{Evnt_{p_c}}(X) \cup T_{S_{P_C}}(X) \cup T_{S_C}(X) \cup T_{Msg_c}(X) \cup T_{Msg_{p_c}}(X)]) \times ([T_{S_{P_C}}(X) \cup T_{S_C}(X) \cup T_{Msg_c}(X) \cup T_{Msg_{p_c}}(X)]) \times (T_{S_*}(X)(X) \cup T_{Msg_*}(X))_{bool}$. The elements of $\mathcal{RC}$ are called rewrite rules, where each rewrite rule[4] is of the form:

$(lcl, |[l_k]|, |[r_l]|, Ccnd(t))$

where:

- lcl is the coordination rule label, and it is an identifier from a specific sort of (coordination) label identifier we denote by L_{CLI}.
- $[l_k] \in [T_{Evnt_{p_c}}(X) \cup T_{S_{P_C}}(X) \cup T_{S_C}(X) \cup T_{Msg_c}(X) \cup T_{Msg_{p_c}}(X)])$. Notice that the event is present in the left-hand of this rule as a mandatory trigger for the rule.
- $[r_l] \in [T_{S_{P_C}}(X) \cup T_{S_C}(X) \cup T_{Msg_c}(X) \cup T_{Msg_{p_c}}(X)])$.
- $Ccnd(lcl) \in (T_{S_*}(x(lcl)) \cup T_{Msg_*}(x(lcl)))_{bool}$. That is, they are boolean (term) expressions constructed from different involved law sorts (i.e. S_C, S_P, S_{Msg_c},$S_{Msg_{P_c}}$, and $S_{Evnt_{P_c}}$). $x(lcl)$ denote the variables occurring in l_k.

Such rewrite rules will be denoted as usual[5]: $lcl : |[l_k]| \Rightarrow |[r_l]|$ `if` $Ccnd(t)$

$\Diamond$

Definition 4.5 (Cor-laws**-entailment inference rules)** Given a Cor-laws rewrite theory $\mathcal{RC}$, we say that $\mathcal{RC}$ entails a sequent $\underline{s} \Rightarrow \underline{s}'$, where $(\underline{s}, \underline{s}')$ are a pair of Cor-laws states, **iff** $\underline{s} \Rightarrow \underline{s}'$ can be obtained by finite (and concurrent) applications of the following rules of deduction.

1. **Reflexivity** : $\forall \quad |[c]| \in ([T_{Evnt_{p_c}}(X) \cup T_{S_{P_C}}(X) \cup T_{S_C}(X) \cup T_{Msg_c}(X) \cup T_{Msg_{p_c}}(X)])$,

$$\overline{|[c]| \Rightarrow |[c]|}$$

[3]As abbreviation any union of the form $[T_{S_i}(\emptyset)] \cup [T_{S_j}(\emptyset)]$ is denoted here instead as $[T_{S_i}(\emptyset) \cup T_{S_j}(\emptyset)]$.

[4]For abbreviation, the star '*' stands for c or P_c.

[5]We use the notation $|[x]|$ to represent the class of elements of x not only modulo the associativity, commutativity with identity (ACI) of $\_\_$, but also modulo eventual axioms from the data level.

2. **Congruence** : $\forall \quad |[c_1]|, |[c'_1]|, |[c_2]|, |[c'_2]|$

$$\frac{|[c_1]| \Rightarrow |[c'_1]| \qquad |[c_2]| \Rightarrow |[c'_2]|}{|[c_1][c_2]| \Rightarrow |[c'_1][c'_2]|}$$

3. **(Concurrent) Replacement**[6]: for each rule

$$t : |[c(x_1, .., x_n)]| \Rightarrow |[c'(x_1, ..., x_n)]| \quad \texttt{if} \quad Ccnd(\bar{x}(t)) \text{ in } R,$$

$$\frac{[w_1] \Rightarrow [w'_1] \ \ldots \ [w_n] \Rightarrow [w'_n] \quad \wedge \quad Ccnd(\bar{w}/\bar{x}(t)) = True}{|[b(\bar{w} \ / \ \bar{x})]| \Rightarrow |[b'(\bar{w'} \ / \ \bar{x})]|}$$

4. **Transitivity** : $\forall \, |[c_1]|, |[c_2]|, |[c_3]|$

$$\frac{|[c_1]| \Rightarrow |[c_2]| \qquad |[c_2]| \Rightarrow |[c_3]|}{|[c_1]| \Rightarrow |[c_3]|}$$

$\Diamond$

4.2 Location Laws in Rewriting Logic

Location Laws can be captured in much the same way. The main difference is that location laws involve reasoning about the distribution context as captured through BT and $REACH$.

A naive way of dealing with these contextual constructs is to include tests in the condition of the rewrite rule associated with a location law exactly as we have described for coordination rules. However, as already stressed, the contextual constructs should not be mixed with the condition for applying the location rule. This is because they are related to the external environment in which the whole application is running and not to specific software/hardware services from the partners.

Our solution consists in extending the labels of the rule so that it can also include the contextual expression that needs to be applied to the rule. We stress here that the idea of extending rewrite rule labels to deal with extra operations is not new; for instance, it has been used in the `ELAN` specification language [vdBMR02, BKKM02] to cope with the control of strategies while performing rules. More precisely, each location rule will be of the form: rl :: `Cxt-exp` : $l \Rightarrow r$, where `Cxt-exp` is the associated contextual expression with the corresponding location law.

Example 4.2 For illustrating the translation of location laws into Maude, we consider the location law relative to the withdrawal at an ATM. The two location interfaces are translated exactly as for coordination interfaces.

[6] In this inference rule $|[b(\bar{w}/\bar{x})]$ denotes the simultaneous substitution of w_i for x_i in b, with $\bar{x}$ representing $x_1, .., x_n$.

```
location interface ATMWdr2-LI
location type ATM
services
      default(), cash():money
      acco():ACCOUNT
      give(n:money)
        post cash() = old cash()-n
events    withdraw(n:money)
end interface
```

```
omod ATMWdr2-LI is
  protecting money  ATM  BOOL.
  subsorts ATOID   BKOID    CSOID < OID .
  class ATMWdr2-LI | cash:money,
      default:money, acco:  ACCOUNT .
  msgs  give :  ATOID   money → Msg .
  evnts  withdraw :  COID   money  → Evnt .
  vars atm :  ATOID .
  vars H : money .
********* Pre- and Post- condition as a rewrite rule.
    rl give(m) ⟨atm|cash : H⟩ ⇒ ⟨atm|cash : H − m⟩ .
endo
```

```
location interface BANKWrd2-LI
location type BANK
services
      internal(n:money, a:ACCOUNT)
      maxatm(a:ACCOUNT): money
end interface
```

```
omod BANKWrd2-LI is
  protecting money  ATM  BOOL.
  subsorts BKOID    AOID < OID .
  class ATMWdr2-LI | maxatm(AOID): money.
  msgs  internal :  ATOID   money   AOID → Msg .
endo
```

```
location law ATMWdr2-LL
locations bank:  BANKWdr2-LI;
      atm:  ATMWdr2-LI
rule :   Withdraw
when atm.withdraw(n)
 and BT(atm,bank)
 with  n ≤ bank.maxatm(atm.acco())
   and n ≤ atm.cash()
 do atm.give(n)
when atm.withdraw(n) and
   ¬ BT(atm,bank)
   and REACH(atm, bank)
   let N=min(atm.default(),n)in
 with N≤atm.cash()
 do atm.give(N)
   mv bank.internal(N, atm.acco()))
end law♢
```

```
omod ATMWdr2-LL is
  using BANKWdr2-LI ATMWdr2-LI .
  protecting Money .
  subsorts CTOID < OID .
  class ATMWdr2-LL | BANKWdr2-LI, ATMWdr2-LI .
  vars H, D, M : money .
  vars atmwdr:  ATMWdr2-LL .
  vars atm:  ATMWdr2-LI.
  vars bank:  BANKWdr2-LI .
********* the location rule as two rewrite rules.
  rl1atm::BT(atm,bank):withdraw(atm, n)
  ⟨atm|cash : H⟩⟨bank|maxatm(acco) : M⟩⟨atmwdr|atm, bank⟩
     ⇒ ⟨atm|cash : H⟩⟨bank|maxatm(acco) : M⟩
     ⟨atmwdr|atm, bank⟩ give(n) if H ≥ n ∧ M > n .
  rl2atm::  ¬ BT(atm,bank) and REACH(atm, bank):
  withdraw(atm, n) ⟨atm|cash : H, default : D⟩
     ⟨bank|maxatm(acco) : M⟩⟨atmwdr|atm, bank⟩
     ⇒ ⟨atm|cash : H, default : D⟩⟨bank|⟩⟨atmwdr|atm, bank⟩
     give(n) ∧ mv(internal(min(D, n), atm)) if min(D, n) ≤ H .
```

As shown in this translation, we have defined a rewrite rule for each case of the applied context. The context expression is written besides the label. ♢

4.2.1 A rewrite theory for Location laws

Up to sort renaming, the definition of the rewrite theories for location laws is similar to the ones proposed for coordination. The only new element resides in the presence of context expressions.

To capture these context expressions, we should first notice that we have two different classes of constructions: (1) The boolean operators such as BT and $REACH$ for 'testing' the context; and (2) the operators related to the actions to be performed depending on these

tests such as mv. In fact, it is the expressions composed of the first class of operators that we are representing in the rule labels. In the following we abstract them using a sort we denote by $Cxtb$. The second class of operators has to be considered just like other messages with a specific sort we denote by $Cxtop$.

Definition 4.6 (Loc-laws-state signature) A location law state signature is defined as a pair $(S_D \cup S_L \cup S_{P_l} \cup S_{Cxtb} \cup S_{Cxtop}, \{Op\}_{S_L} \cup \{Op\}_{S_{P_l}})$ with:

- S_D is a set of (data) sorts with at least: $\{bool, LOId, PlOId, AId, Value\} \subset S_D$. To allow object-attribute values, OId is defined as a subsort of $Value$ (i.e. $OId < Value$).
- S_L is a set of location sorts (different from S_D), which we assume contains at least one sort.
- S_{P_l} is a set of partners (i.e. interfaces) sorts (different from S_D and S_L). We assume S_{P_l} contains at least one sort.
- $\{Op\}_{S_L}$ is a set of 'location state' operations indexed by $LOId \times (PlOId \times S_{P_l})^n \times (AId \times Value)^+ \times S_L$. More precisely, with each coordination sort from S_L a 'location state' operation is associated with n partners ($n \geq 1$).
- $\{Op\}_{S_{Cxtb}}$ is a set of boolean context operations indexed by $LOId^n \times BOOL$.
- $\{Op\}_{S_{Cxtop}}$ is a set of context operations indexed by $(LOId^n \times S_D) \times S_{Cxtop}$.
- $\{Op\}_{S_{P_l}}$ is a set of 'partner state' operations indexed by $PlOId \times (AId \times Value)^+ \times S_{P_l}$. More precisely, with each partner sort from S_{P_l} a 'partner state' operation is associated. $\Diamond$

On the basis of this notion of Loc-laws state signature, a Loc-laws template signature is defined by extending it by message and event sorts and operations.

Definition 4.7 (Loc-laws-template signature) A template signature is defined as a pair $(S_D \cup S_L \cup S_{P_l} \cup S_{Cxtb} \cup S_{Cxtop} \cup S_{Msg_l} \cup S_{Msg_{p_l}} \cup S_{Evnt_{p_l}}, \{Op\}_{S_L} \cup \{Op\}_{S_{Msg_p}} \cup \{Op\}_{S_{Msg_{p_l}}} \{Op\}_{S_{Evnt_{p_l}}})$ with:

- $(S_D \cup S_O \cup S_L \cup S_{Cxtb} \cup S_{Cxtop}, \{Op\}_{S_P} \cup \{Op\}_{S_{P_l}})$ is a Loc-laws state signature as defined above.
- $S_{Msg_l} \cup S_{Msg_{p_l}}$ are a disjoint sets of 'message generator' sorts. They correspond to the operation sorts defined in the location laws and interfaces respectively.
- $\{Op\}_{S_{Msg_l}}$ (resp. $\{Op\}_{S_{Msg_{p_l}}}$), is a set of message operations. That is, they are operations indexed by $LOId \times S_D^* \times S_{Msg_l}$ (resp. $PlOId \times S_D^* \times S_{Msg_{p_l}}$).
- $\{Op\}_{S_{Evnt_{p_l}}}$, is a set of event operations. That is, they are operations indexed by $PlOId \times S_D^* \times S_{Evnt_{p_l}}$. $\Diamond$

Template specification. Given a Loc-laws-template signature denoted by LTS that captures the structural aspects of location laws, its behaviour is constructed by associating it what we call a Loc-laws-rule system $\mathcal{RL}$ and corresponding Loc-laws-rewrite theory $\mathcal{RT}_{\mathcal{L}}$. —leading to the notion of Loc-laws template specification that we denote by $SPL = \prec LTS, \mathcal{RL} \succ_{\mathcal{RT}_{\mathcal{L}}}$.

Notation 4.2

1. We denote by $T_{S_L}(X_{Value})$ (resp. $T_{S_{P_l}}(X_{Value})$) the set of algebraic terms of sort(s) S_L (resp. S_{P_l}); where X_{Value} is an s_atp_i-(resp. s_atpl_i) indexed set of variables; with

s_atp_i and s_atpl_i the specific sorts of attributes associated with each location or partner state sort. That is, for each (location or partner) attribute sort we assume given a set of variables. In the sequel we just use X as a union of all these sets of variables. By $T_{S_P}(\emptyset)$ (resp. $T_{P_l}(\emptyset)$) we denote their associated (ground) object terms.

2. Similarly, we will denote by $T_{Msg_L}(X)$ (resp. $T_{Msg_{P_l}}(X)$ and $T_{Evnt_{P_l}}(X)$) the set of algebraic terms of sort(s) S_{Msg_L} (resp. $S_{Msg_{P_l}}$ and $S_{Evnt_{P_l}}$).

3. $T_{Cxtop}(X)$ (resp. $T_{Cxtb}(X)$ denotes the terms generated from the context operations (resp. states).

4. To represent different multi-sets of terms (in states or rules), we denote by $MT_{S_L}(X)$ and (resp. $MT_{Msg_L}(X)$) the multiset of terms over $T_{S_L}(X)$ (resp. $T_{Msg_L}(X)$), with the 'blank' binary $\_\_$ as union (associative and commutative) operation, and $\emptyset_M$ as the identity element. The two multiset forms will be subsequently referenced by $[T_{S_L}(X)]$ and $[T_{Msg_L}(X)]$.

5. In the same way we denote by $MT_{P_L}(X)$ and (resp. $MT_{Msg_{P_l}}(X)$ and $MT_{Evnt_{P_l}}(X)$) the multiset of terms over $T_{S_{P_l}}(X)$ (resp. $T_{Msg_{P_l}}(X)$ and $T_{Evnt_{P_l}}(X)$), with the 'blank' binary $\_\_$ as union (associative and commutative) operation, and $\emptyset_M$ as the identity element. The three multiset forms will be subsequently referenced by $[T_{P_L}(X)]$, $[T_{Msg_{P_l}}(X)]$ and $[T_{Evnt_{P_l}}(X)]$.

Definition 4.8 (Loc-laws-states) A Loc-laws state is an element of $[T_{S_P}(\emptyset) \cup T_{S_{P_l}}(\emptyset) \cup T_{Msg_l}(\emptyset) \cup T_{Msg_{p_l}}(\emptyset) \cup T_{Evnt_{p_l}}(\emptyset)] \cup [T_{Cxtop}(\emptyset)]$. $\Diamond$

Definition 4.9 (Loc-laws rewrite theory) Given a Loc-laws template-signature as presented above, an Loc-laws rewrite theory is a set of sixtuples $\mathcal{RL} \subset L_{CLI} \times [T_{Cxtb}(X)] \times ([T_{Evnt_{p_l}}(X) \cup T_{S_{P_L}}(X) \cup T_{S_L}(X) \cup T_{Msg_l}(X) \cup T_{Msg_{p_l}}(X)]) \times ([T_{S_{P_L}}(X) \cup T_{S_L}(X) \cup T_{Msg_l}(X) \cup T_{Msg_{p_l}}(X) \cup T_{Cxtop}(X)]) \times (T_{S_*}(X) \cup T_{Msg_*}(X))_{bool}$. The elements of $\mathcal{RL}$ are called rewrite rules, where each rewrite rule is of the form:

$(lwl, |[Cx_i]|, (|[l_k]|), (|[r_l]|), Lcnd(t))$

where:

- lwl is the location rule label, and it is an identifier from a specific sort of (location) label identifier we denote by L_{CLI}.
- $|[Cx_i]| \in [T_{Cxtb_L}(X)$ is the location law context associated to the location law.
- $[l_k] \in [T_{Evnt_{p_l}}(X) \cup T_{S_{P_L}}(X) \cup T_{S_L}(X) \cup T_{Msg_p}(X) \cup T_{Msg_{p_l}}(X)])$.
- $[r_l] \in [T_{S_{P_L}}(X) \cup T_{S_L}(X) \cup T_{Msg_l}(X) \cup T_{Msg_{p_l}}(X) \cup T_{Cxtop_L}(X)]$.
- $Lcnd(lwl) \in (T_{S_*}(x(lcl)) \cup T_{Msg_*}(x(lcl)))_{bool}$. That is, they are boolean (term) expressions constructed from different involved law sorts (i.e. S_L, $S_{P_l}S_{Msg_l}$,$S_{Msg_{P_l}}$, and $S_{Evnt_{P_l}}$). $x(lwl)$ denote the variables occurring in l_k. "*" stands for "l" and "P_l"

Such rewrite rules will be denoted as follows:

$$lwl : |[Cx_i]| : |[l_k]| \Rightarrow |[r_l]| \quad \texttt{if} \quad Lcnd(t)$$

$\Diamond$

Definition 4.10 (LOC-LAWS**-entailment inference rules)** Given a LOC-LAWS rewrite theory $\mathcal{LR}$, we say that $\mathcal{LR}$ entails a sequent $\underline{s} \Rightarrow \underline{s}'$, where $(\underline{s}, \underline{s}')$ are a pair of LOC-LAWS states, **iff** $\underline{s} \Rightarrow \underline{s}'$ can be obtained by finite (and concurrent) applications of the following rules of deduction[7].

(Concurrent) Replacement

for each rule

$$rlw :: |[Cxb(loc_1, .., loc_i)]| : \ |[l(x_1, .., x_n)]| \Rightarrow |[l'(x_1, ..., x_m)]| \quad \texttt{if} \quad Lcnd(\bar{x}(t))$$

in $\mathcal{RL}$,

$$\frac{[w_1] \Rightarrow [w'_1] \ \ldots \ [w_n] \Rightarrow [w'_n] \quad \wedge \quad Cxb(\bar{w}/\bar{x}(rlw)) = True \wedge \quad Lcnd(\bar{w}/\bar{x}(t)) = True}{|[l(\bar{w} \ / \ \bar{x})]| \Rightarrow |[l'(\bar{w'} \ / \ \bar{x})]|}$$

$\Diamond$

5 Compositional coordination/location validation

In the previous section we proposed a complete interpretation of location and coordination laws in Rewriting Logic. The purpose of this second step is to deal with more than one coordination (resp. location) law when validating the system. For that aim, we start by introducing an adequate (data) representation for both coordination/location states and rules, which facilitates the composition of two or more laws.

That is, given a coordination (resp. location) state denoted by $\underline{s}$ or a rule $rl : l \Rightarrow r$, the aim is to reorganize this state or rule in such a way that the corresponding multi-terms for each partner can be explicitly distinguished. We achieve this by associating with each part of such state a partner name or coordination name (i.e. as an identifier).

Before detailing this technical transformation and discussing its benefits, let us explain what we want to achieve through the following coordination rewrite rule associated with the withdrawal coordination law already discussed.

Example 5.1 We start by recalling that this rule takes the following form:

rlw `withdraw(cust, acco, n)` $\langle acco|bal : B\rangle\langle cust|CsName : Nm\rangle\langle sdwdr|acco, cust\rangle$
$\Rightarrow \langle acco|bal : B\rangle\langle cust|CsName : Nm\rangle\langle sdwdr|acco, cust\rangle$
$debit(cust, acco, n)$ **if** $B \geq n \wedge owns(cust, acco) = true$

In this rule the following multi-term $\texttt{withdraw}(\texttt{cust}, \texttt{acco}, \texttt{n})\langle cust|CsName : Nm\rangle$ is from the customer partner, whereas $\langle acco|bal : B\rangle debit(cust, acco, n)$ is from the account partner. Finally, $\langle sdwdr|acco, cust\rangle$ is from the coordination class.

The idea is to associate with each of these parts the corresponding partners using a notation like `(partner-Name, multi-term)` and then compose them using an associative/commutative operator $\otimes$. In this way, any left hand-side, right-hand side or state will be represented as $(pc_1, mt_1) \otimes ... \otimes (pc_i, mt_i)$, with pc_i partner names and mt_i their corresponding multi-terms.

Following these guidelines, our rewrite rule should now look as follows:

[7] The three first deduction rules are just like the one for coordination and are omitted here.

$$
\begin{aligned}
&\mathbf{rlw} : (CustomerP, \texttt{withdraw}(\texttt{cust}, \texttt{acco}, \texttt{n})\langle cust|CsName : Nm\rangle)\otimes \\
&\quad (AccountP, \langle acco|bal : B\rangle) \otimes (WdrsdP, \langle sdwdr|acco, cust\rangle) \\
&\qquad \Rightarrow (CustomerP, \langle cust|CsName : Nm\rangle) \otimes (AccountP, \langle acco|bal : B\rangle debit(cust, acco, n))\otimes \\
&\qquad (WdrsdP, \langle sdwdr|acco, cust\rangle) \\
&\qquad \textbf{if } B \geq n \wedge owns(cust, acco) = true
\end{aligned}
$$

In this derived rule, `CustomerP, AccounP` and `WdrsdP` are just (partner/contract) class or component identifiers, used for factorizing different parts of this rule. These ideas are made more formal in the following. ♢

5.1 Compositional Coordination states and rules

First, we define the multi-sets generated by the operator $\otimes$ through the following notations.

Notation 5.1

1. We denote by $BT_{Crd}(X)$ (resp. $BT_{Prt_c}(X)$) the multi-set form over over $C_c \times ([T_{S_C}(X)] \cup [T_{Msg_C}(X)])$ (resp. $P_c \times ([T_{S_{P_c}}(X)] \cup [T_{Msg_{P_c}}(X)] \cup [T_{Evnt_{P_c}}(X)])$). Both mutisets are to be generated by the union operator denoted by $\otimes$ (with $\emptyset_B$ as identity).

2. As precisely defined below, $\{C_c, P_c\}$ denotes a set of coordination and partners identifiers.

3. Elements of both multiset forms will be subsequently abbreviated by $[T_C(X)]_\otimes$ and $[T_{P_c}(X)]_\otimes$ respectively.

Definition 5.1 (Compositional Cor-laws**-template specification)** Given a Cor-laws-template specification, a compositional Cor-laws specification is constructively defined from it as a structure $(C_c, P_c, s_c, tr_p, tr_c)$ where:

- C_c (resp.P_c) is a set of coordination (resp. partner) identifiers, such that $|C_c| = |S_C|$ (resp. $|P_c| = |S_{P_c}|$). That is, the number of coordination (resp. partner) identifiers is exactly the cardinality of sorts in S_C (resp. S_{P_c}).
- $s_c : C_c \longrightarrow S_C \times \{S_{Msg_c}\}$ is a bijection associating with each coordination identifier in C_c a corresponding sort from S_C with all related message sorts in S_{Msg_c}. That is, s_c allows type checking of any (correct) sub-configuration associated with a given coordination identifier.
- $s_{p_c} : P_c \longrightarrow S_{P_c} \times \{S_{Msg_{P_c}} \cup S_{Evnt_{P_c}}\}$ is a bijection associating with each coordination identifier in P_c a corresponding sort from S_{P_c} with all ites corresponding message and event sorts in $\{S_{Msg_{P_c}} \cup S_{Evnt_{P_c}}\}$.
- $tr_c : [T_{S_C}(X) \cup T_{S_{Msg_c}}(X)] \longrightarrow [C_c \times ([T_{S_C}(X)] \cup [T_{S_{Msg_c}}(X)])]_\otimes$. That is, if c_i is a multiset term from $[T_{S_C}(X)] \cup [T_{S_{Msg_c}}(X)]$, then $tr_c(mt_i)$ allows transforming it into another multiset term of the form $\underset{i}{\otimes}(c_i, mt_i)$ over $[C_c \times ([T_{S_C}(X)] \cup [T_{S_{Msg_c}}(X)]]_\otimes$. c_i denotes any coordination identifier referencing the coordination class.
- $tr_{p_c} : [T_{S_{p_c}}(X)] \cup [T_{S_{Msg_{p_c}}}(X)] \cup [T_{S_{Evnt_{p_c}}}(X)] \longrightarrow [P_c \times ([T_{S_{p_c}}(X)] \cup [T_{S_{Msg_{p_c}}}(X)] \cup [T_{S_{Evnt_{p_c}}}(X)])]_\otimes$. That is, if nt_i is a multiset term from $[T_{S_{p_c}}(X)] \cup [T_{S_{Msg_{p_c}}}(X)] \cup [T_{S_{Evnt_{p_c}}}(X)]$, then $tr_c(mt_i)$ allows transforming it into another multiset term of the form $\underset{i}{\otimes}(p_i, nt_i)$ over $[C_c \times ([T_{S_{p_c}}(X)] \cup [T_{S_{Msg_{p_c}}}(X)] \cup [T_{S_{Evnt_{p_c}}}(X)])]_\otimes$. p_i denotes any partner identifier referencing a partner class. ♢

Remark 5.1 *These transformations towards a compositional coordination deal with the two aspects related to configurations (i.e. states and rules). They can be made more readable in the form of inference rules as follows:*

***Coordination states transformation**: $\forall\, c_k \in C_c, \forall\, [n_{c_k}] \in [T_{S_C}(\emptyset) \cup T_{Msg_c}(\emptyset)]$, and $\forall\, p_{c_j} \in P_c, \forall\, [m_{pc_j}] \in [T_{S_{P_c}}(\emptyset) \cup T_{Msg_{P_c}}(\emptyset) \cup T_{Evnt_{P_c}}(\emptyset)]$; $j \in \{1, .., |S_{p_c}|\}$.*

$$\frac{\underset{j=1}{\overset{--}{}}[m_{pc_j}]\ [n_{c_k}]}{\underset{j=1}{\otimes}(pc_j, [m_{pc_j}]) \otimes (c_k, [n_{c_k}])}$$

***Coordination rules transformation**: $\forall\, c_k \in C_c, \forall\, [ln_{c_k}], [rn_{c_k}] \in [T_{S_C}(X) \cup T_{Msg_c}(X)]$, and $\forall\, p_{c_j} \in P_c, \forall\, [lm_{pc_j}], [rm_{pc_j}] \in [T_{S_{P_c}}(X) \cup T_{Msg_{P_c}}(X) \cup T_{Evnt_{P_c}}(X)]$; $j \in \{1, .., |S_{p_c}|\}$.*

$$\frac{\forall\, lc_k : \underset{j=1}{\overset{--}{}}[lm_{pc_j}]\ [ln_{c_k}] \Rightarrow \underset{j=1}{\overset{--}{}}[rm_{pc_j}]\ [rn_{c_k}]\ \textbf{\textit{if}}\ Ccnd\ \in \mathcal{RC}}{lc_k : \underset{j=1}{\otimes}(pc_j, [lm_{pc_j}]) \otimes (c_k, [ln_{c_k}]) \Rightarrow \underset{j=1}{\otimes}(pc_j, [rm_{pc_j}]) \otimes (c_k, [rn_{c_k}])\ \textbf{\textit{if}}\ Ccnd}$$

$\Diamond$

In this way, several compositional coordination states can be merged in a straightforward way, allowing to analyze inter-coordination configurations. This merging can be realized by the following inference rule. We note that the compositional rules (from different laws) remain unchanged yet directly applied to these merged states.

Definition 5.2 (Compositional Cor-laws**-state)** Given some compositional Cor-laws-states), their merging could be achieved through the following inference rule:

Compositional states transformation: $\forall\, c_i \in C_c, \forall\, [n_{c_i}] \in [T_{S_C}(X) \cup T_{Msg_c}(X)]$, and $\forall\, p_{c_{ij}} \in P_c, \forall\, [m_{pc_{ij}}] \in [T_{S_{P_c}}(X) \cup T_{Msg_{P_c}}(X) \cup T_{Evnt_{P_c}}(X)]$; $i \in \{1, .., |S_C|\}$ $j \in \{1, .., |S_{p_c}|\}$.

$$\frac{\bigcup_{i=1}(c_i, [n_{c_i}]) \underset{j=1}{\otimes}(p_{c_{ij}}, [m_{pc_{ij}}])}{\underset{i=1}{\otimes}(c_i, [n_{c_i}]) \underset{i,j=1}{\otimes}(p_{c_{ij}}, [m_{pc_{ij}}])}$$

$\Diamond$

Remark 5.2

1. *We denote by $[tr_c[T_{s_c(c)}(X)]]_\otimes$ (resp. $[tr_p[T_{p_c(p)}(X)]]_\otimes$) any multi-set term generated from the above transformation when applied to a specific coordination class $c \in C_c$ (resp. partner $p \in P_c$).*

2. *With the above transformation, the rules now take the form:*
   $t : (|[\overset{n_p}{\otimes}(p_k, [l_k])]_\otimes|) \Rightarrow (|[\overset{n_q}{\otimes}(q_l, [r_l])]_\otimes|)$ $\quad$ ***if*** $\quad Ccnd(t)$ $\quad$ *With $p_k, q_l \in \{C_c, P_c\}$.*

3. *In the same way the compositional* Cor-laws*-state takes the form: $[C_c \times ([T_{S_C}(\emptyset) \cup T_{Msg_c}(\emptyset)])]_\otimes \cup [P_c \times ([T_{S_{p_c}}(\emptyset) \cup T_{Msg_{p_c}}(\emptyset) \cup T_{Evnt_{p_c}}(c)])]_\otimes$. More precisely, by denoting such state by $\underline{s}$, it can be written as follows: $\underline{s} = \underset{p_i \in \{C_c, P_c\}}{\otimes}(p_i, [T_{s(p_i)}(\emptyset)])$. With s denoting either s_c or s_{p_c} accordingly to the identifier p_i.* $\Diamond$

5.1.1 Compositional Cor-laws-rewrite theory

To reason about this new form of rules and states, we need an appropriate rewrite theory. Besides the reflexivity, congruence, replacement and transitivity, the main additional inference rule that should be present in such a rewrite theory is the one to allow a maximum of parallelism with respect to the multiset operator $\otimes$. Intuitively, this is achieved using an inference that allows the following: $(p_1, mt_1) \otimes (p_1, mt_2) = (p_1, mt_1\ mt_2)$. That is, we should be able to split/recombine the state or the rule when required.

Definition 5.3 (Compositional Cor-laws-rewrite theory) The compositional Cor-laws-rewrite theory takes the following form:

1. **Reflexivity** : $\forall \quad |[b]_\otimes| \in [C_c \times ([T_{S_C}(X) \cup T_{Msg_c}(X)])]_\otimes \cup [P_c \times ([T_{S_{p_c}}(X) \cup T_{Msg_{p_c}}(X) \cup T_{Evnt_{p_c}}(X)])]_\otimes$,

$$\frac{}{|[b]_\otimes| \Rightarrow |[b]_\otimes|}$$

2. **Congruence** : $\forall \quad |[b_1]_\otimes|, |[b'_1]_\otimes|, |[b_2]_\otimes|, |[b'_2]_\otimes|$

$$\frac{|[b_1]_\otimes| \Rightarrow |[b'_1]_\otimes| \qquad |[b_2]_\otimes| \Rightarrow |[b'_2]_\otimes|}{|[b_1]_\otimes \otimes [b_2]_\otimes| \Rightarrow |[b'_1]_\otimes \otimes [b'_2]_\otimes|}$$

3. **(Concurrent) Replacement**: for each rule

$$t : |[b(x_1, .., x_n)]_\otimes| \Rightarrow |[b'(x_1, ..., x_n)]_\otimes| \quad \texttt{if} \quad Ccnd(\bar{x}(t)) \text{ in } R,$$

$$\frac{[w_1] \Rightarrow [w'_1] \ \ldots \ [w_n] \Rightarrow [w'_n] \quad \wedge \quad Ccnd(\bar{w}/\bar{x}(t)) = True}{|[b(\bar{w}\ /\ \bar{x})]_\otimes| \Rightarrow |[b'(\bar{w'}\ /\ \bar{x})]_\otimes|}$$

4. **Transitivity** : $\forall\, |[b_1]_\otimes|, |[b_2]_\otimes|, |[b_3]_\otimes|$

$$\frac{|[b_1]_\otimes| \Rightarrow |[b_2]_\otimes| \qquad |[b_2]_\otimes| \Rightarrow |[b_3]_\otimes|}{|[b_1]_\otimes| \Rightarrow |[b_3]_\otimes|}$$

5. **State Splitting and Recombining**: $\forall\, cp \in \{C_c, P_c\}, \forall\, [n] \in [T_{s(cp)}]$, and $\forall\, [m_i] \in [T_{s(cp)}]$; $i \in \{1, .., n_{cp}\}$.

$$\frac{|[n]| = |[\overset{n_{cp}}{\underset{i=1}{=}} m_i]|}{|[(cp, [n])]_\otimes| = |[\overset{n_{cp}}{\underset{i=1}{\otimes}} (cp, [m_i])]_\otimes|}$$

$$\frac{\forall\, cp \in \{C_c, P_c\}\ ;\ \forall\, [n], [n'] \in [T_{s(cp)}]}{|[(cp, [n])]_\otimes| \otimes |[(cp, [n'])]_\otimes| = |[(cp, [n][n'])]_\otimes|}$$

6. **Identity**: $\forall\, cp \in \{C_c, P_c\}$

$$\frac{True}{(cp, \emptyset_M) = \emptyset_B}$$

$\Diamond$

5.2 Compositional Location Laws in Rewriting Logic

In the same way, we have to transform a given LOC-LAWS-specification into a compositional one. The same constructions have to be applied up-to renaming (e.g. L_l instead of C_c, P_l instead of P_c, S_P instead of S_C, etc).

6 Integration of concerns in Rewriting Logic

The purpose of this part is to integrate different concerns bringing together their compositional specifications. That is, according to the informal description of integration given in the second section, we have to unify or synchronize all rules dealing with the same event from both concerns. In our case, we have also to bring together the corresponding (coordination and location) compositional states.

Informally, like the process of deriving compositional coordination and location laws, we have to combine different location and coordination partners with their associated multi-terms. In the following example we shed more light of this integration process, before formalizing it using adequate inference rules.

Example 6.1 First let us consider the above resulted compositional rewrite rule from the coordination withdrawal coordination law:

$$\begin{array}{l}
\textbf{rlw} : (CustomerP, \texttt{withdraw}(\texttt{cust}, \texttt{acco}, \texttt{n})\langle cust|CsName : Nm\rangle)\otimes \\
\quad (AccountP, \langle acco|bal : B\rangle) \otimes (WdrsdP, \langle sdwdr|acco, cust\rangle) \\
\qquad \Rightarrow (CustomerP, \langle cust|CsName : Nm\rangle) \otimes (AccountP, \langle acco|bal : B\rangle debit(cust, acco, n))\otimes \\
\qquad (WdrsdP, \langle sdwdr|acco, cust\rangle) \\
\qquad \textbf{if } B \geq n \wedge owns(cust, acco) = true
\end{array}$$

By applying the same transformations related to location laws, we can result in the following compositional rule from ATM-withdrawal location rule:

$$\begin{array}{l}
\textbf{rl1atm}::\textbf{BT}(atm, bank) : (ATMP, \texttt{withdraw}(\texttt{atm}, \texttt{n})\langle atm|cash : H\rangle)\otimes \\
\quad (BANKP, \langle bank|maxatm(acco) : M\rangle) \otimes (ATMwdrP, \langle atmwdr|atm, bank\rangle) \\
\quad \Rightarrow (ATMP, \langle atm|cash : H\rangle\ give(n)) \otimes (BANKP, \langle bank|maxatm(acco) : M\rangle)\otimes \\
\quad (ATMwdrP, \langle atmwdr|atm, bank\rangle) \quad \textbf{if} \quad H \geq n \wedge M > n \ .
\end{array}$$

From these two compositional rules, it become obvious that we have to regroup: the multi-terms related to `CustomerP` with those related to `ATMP`, the multi-terms related to `AccounP` with those related to `BANKP`. Finally, the multi-terms related to the coordination and location parts have to be merged (i.e. `WdrsdP@ATMwdrP`). More precisely, the aim is to speak about `CustomerP@ATMP` and `AccounP@BANKP` as unique components. The withdraw event has to be present just one time, but including all parameters present in both rules (we refer to this operation as events unification).

All in all, the final rule takes the following form:

$$\begin{array}{l}
\textbf{rl1}_{cs@atm}::\textbf{BT}(atm, bank): \\
\quad (CustomerP@ATMP, \texttt{withdraw}(\texttt{atm}, \texttt{cust}, \texttt{acco}, \texttt{n})\langle cust|CsName : Nm\rangle\ \langle atm|cash : H\rangle)\otimes \\
\quad (AccountP@BANKP, \langle acco|bal : B\rangle\ \langle bank|maxatm(acco) : M\rangle)\otimes \\
\qquad (WdrsdP@ATMwdrP, \langle sdwdr|acco, cust\rangle\langle atmwdr|atm, bank\rangle) \\
\qquad \Rightarrow (CustomerP@ATMP, \langle cust|CsName : Nm\rangle\ \langle atm|cash : H\rangle)\otimes
\end{array}$$

$(AccountP@BANKP, \texttt{debit(cust, acco, n)}\langle acco|bal : B\rangle\ \langle bank|maxatm(acco) : M\rangle)\langle\otimes$
$(WdrsdP@ATMwdrP, \langle sdwdr|acco, cust\rangle\langle atmwdr|atm, bank\rangle)$
if $B \geq n \wedge owns(cust, acco) = true$

An illustration of this binding 'coordination@location' partners and contracts could also be understood from the figure 2 below. In fact, in contrast to the integration of concerns depicted in Figure 1, this integration is done without the need of superposition as we are considering the interfaces are object classes. ♢

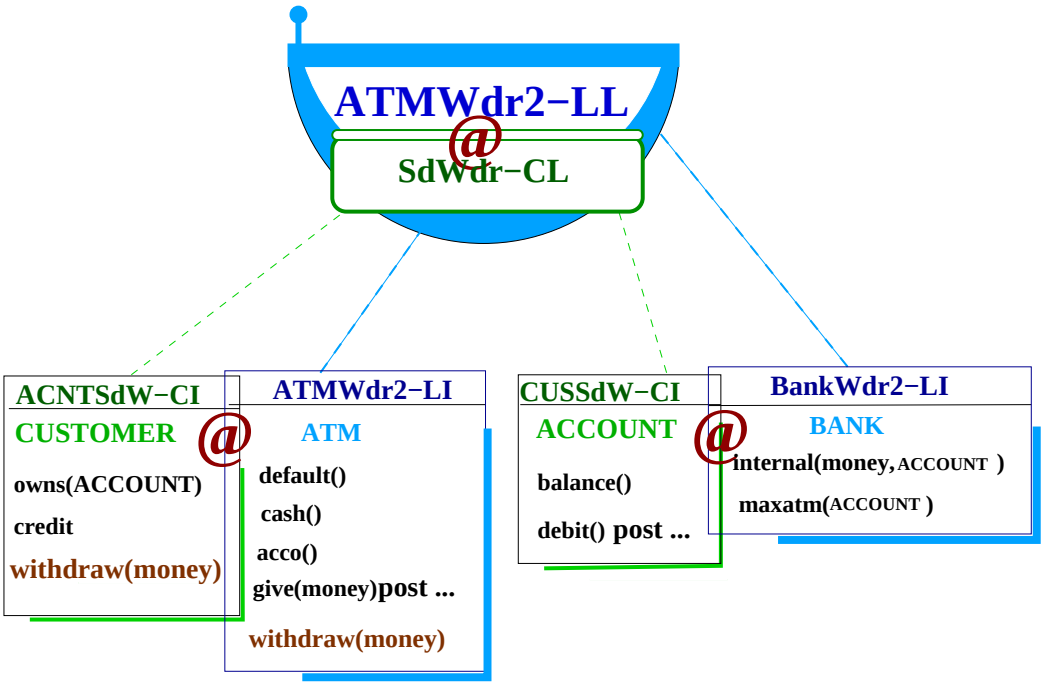


Figure 2: The Integration of concerns at an abstract level.

6.1 Integration of concerns : Formal foundation

Definition 6.1 (Integration of Compositional COR-LAWS**- and** LOC-LAWS**-specification)** Given a compositional COR-LAWS specification $(C_c, P_c, s_c, tr_{pc}, tr_c)$ and compositional LOC-LAWS specification $(L_l, P_l, s_p, tr_{pl}, tr_l)$, their compositional integration is constructed as a structure of the form $(C_c@L_l, P_c@P_l, s_{c@l}, tr_{pc@pl}, tr_{c@l}, S_{Evnt_{c@l}}, tr_{ev})$

- $C_c@L_l$ (resp. $P_c@P_l$) is a new set of 'located' coordinations (resp. located partner) identifiers reflecting the merging of (location and coordination) interfaces and laws. We impose that $|C_c@L_l| \leq |S_C| \times |S_L|$) (resp. $|P_c@P_l| \leq |S_{P_c}| \times |S_{P_l}|$). That is, the cardinality of $C_c@L_l$ (resp. $P_c@P_l$) should not go beyond the product of their constituents (i.e. $|S_C| \times |S_L|$ and $|S_{P_c}| \times |S_{P_l}|$).
- $s_{c@l} : C_c \times L_l \longrightarrow S_C \times S_L \times \{S_{Msg_c} \cup S_{Msg_l}\}$ is a bijection associating with each located-coordination identifier in $C_c \times L_l$ a corresponding sort from $S_C \times S_L \times \{S_{Msg_c} \cup S_{Msg_l}\}$.
- $tr_{Ev} : S_{Evnt_{pc}} \times S_{Evnt_{pl}} \longrightarrow S_{Evnt_{c@l}}$ is a bijection associating with each couple of event sorts from coordination and location partners a join event sort.
- $tr_{c@l} : [C_c \times ([T_{S_C}(X)] \cup [T_{S_{Msg_c}}(X)])]_\otimes \cup [L_l \times ([T_{S_P}(X)] \cup [T_{S_{Msg_p}}(X)])]_\otimes \longrightarrow [C_c@L_l \times ([T_{S_C}(X)] \cup [T_{S_{Msg_c}}(X)] \cup [T_{S_P}(X)] \cup [T_{S_{Msg_p}}(X)])]_\otimes$. That is, if we have two multiset

(c_i, n_i) and (l_i, m_i), the $tr_{c@l}((c_i, n_i), (l_i, m_i))$ results in $(c_i@l_i, n_i\ m_i)$.

- $tr_{pc@pl} : [P_c \times ([T_{S_{P_c}}(X)] \cup [T_{S_{Msg_{pc}}}(X)] \cup [T_{S_{Evnt_{pc}}}(X)])]_\otimes \cup [P_l \times ([T_{S_{P_l}}(X)] \cup [T_{S_{Msg_{pl}}}(X)] \cup [T_{S_{Evnt_{pl}}}(X)])]_\otimes \longrightarrow [P_c@P_l \times ([T_{S_{P_c}}(X)] \cup [T_{S_{P_l}}(X)][T_{S_{Msg_{p_c}}}(X)] \cup [T_{S_{Msg_{p_l}}}(X)] \cup [T_{S_{Evnt_{p@l}}}(X)])]_\otimes$. That is, $tr_{pc@pl}$, merges the coordination partners parts with those of the location ones. Notice the unification of events into a one single part. $\Diamond$

Remark 6.1 *These transformations are only applied when the context is true: otherwise the transformation concerns only the location part. These transformations can also be expressed in terms of inference rules as follows:*

1. ***Integration of coordination/location states:***

   $\forall\, c_j \in C_c, \forall\, [n_{c_j}] \in [T_{S_C}(\emptyset) \cup T_{Msg_c}(\emptyset)]$, *and* $\forall\, p_{c_{ij}} \in P_c, \forall\, [m_{p_{c_{ij}}}] \in [T_{S_{P_c}}(\emptyset) \cup T_{Msg_{P_c}}(\emptyset)], \forall\, [ev_{c_{ij}}] \in [T_{Evnt_{P_c}}(\emptyset)]$;
   $\forall\, l_v \in L_l, \forall\, [s_{l_v}] \in [T_{S_L}(\emptyset) \cup T_{Msg_l}(\emptyset)]$, *and* $\forall\, p_{l_{uv}} \in P_l, \forall\, [r_{p_{l_{uv}}}] \in [T_{S_{P_l}}(\emptyset) \cup T_{Msg_{P_l}}(\emptyset)], \forall\, [ev_{l_{uv}}] \in [T_{Evnt_{P_l}}(\emptyset)]$;

$$\frac{\begin{array}{c}\bigotimes_{i,j=1} (pc_{ij}, [ev_{c_{ij}}]\ [m_{pc_{ij}}]) \otimes (c_j, [n_{c_j}]) \in Conf(C^{cp}_{st}) \\ \bigotimes_{u,v=1} (p_{l_{uv}}, [ev_{l_{uv}}][r_{p_{l_{uv}}}]) \otimes (l_v, [s_{l_v}]) \in Conf(L^{cp}_{st})\end{array}}{\bigotimes_{x,y=1} (p_{c@lc_{xy}}, [ev_{c@lc_{xy}}]\ [m_{pc_{xy}}]\ [r_{pl_{xy}}]) \otimes (c@lc_y, [n_{c_y}]\ [s_{l_y}]) \in Conf(C@L^{cp}_{st})}$$

2. ***Integration of coordination/location rules:***

   $\forall\, c_k \in C_c, \forall\, [n_{c_k}], [n'_{c_k}] \in [T_{S_C}(X) \cup T_{Msg_c}(X)]$, *and* $\forall\, p_{c_i} \in P_c, \forall\, [m_{p_{c_i}}], [m'_{p_{c_i}}] \in [T_{S_{P_c}}(X) \cup T_{Msg_{P_c}}(X)], \forall\, [ev_{p_{c_i}}] \in [T_{Evnt_{P_c}}(X)]$; $i \in \{1, .., |S_{P_c}|\}$.
   $\forall\, l_o \in L_l, \forall\, [s_{l_o}], [s'_{l_o}] \in [T_{S_L}(X) \cup T_{Msg_l}(X)]$, *and* $\forall\, p_{l_j} \in P_l, \forall\, [r_{p_{l_j}}], [r'_{p_{l_j}}] \in [T_{S_{P_l}}(X) \cup T_{Msg_{P_l}}(X) \cup T_{Evnt_{P_l}}(X)]$; $j \in \{1, .., |S_{P_l}|\}$.

   $\forall rl_{c_{i_1}} \in \mathcal{RC}, \forall rl_{l_{j_1}} \in \mathcal{RL}$

$$\frac{\begin{array}{c} rl_{c_{i_1}} : \bigotimes_{i=1} (p_{c_i}, [ev_{c_i}]\ [m_{p_{c_i}}]) \otimes (c_k, [n_{c_k}]) \Rightarrow \bigotimes_{i=1} (p_{c_i}, [m'_{p_{c_{i_1}}}]) \otimes (c_k, [n'_{c_k}])\ \textit{if Ccnd} \\ rl_{l_{j_1}} : Cxtb_{l_{j_1}} : \bigotimes_{j=1} (p_{l_j}, [ev_{l_j}][r_{p_{l_j}}]) \otimes (l_o, [s_{l_o}]) \Rightarrow \bigotimes_{j=1} (p_{l_j}, [r'_{p_{l_j}}]) \otimes (l_o, [s'_{l_o}])\ \textit{if Lcnd}\end{array}}{\begin{array}{c} rl_{c@l_{k_1}} :: Cxtb_{l_{k_1}} : \bigotimes_{j=1} (p_{c@lc_j}, [ev_{c@lc_j}]\ [m_{pc_j}]\ [r_{pl_j}]) \otimes (c@lc_k, [n_{c_k}]\ [s_{l_o}]) \\ \Rightarrow \bigotimes_{j=1} (p_{c@lc_j}, [m'_{pc_j}]\ [s'_{lc_o}]) \otimes (c@lc_k, [n'_{c_k}]\ [t'_{lc_o}])\ \textit{if Ccnd} \wedge \textit{Lcond}\end{array}}$$

$\Diamond$

The corresponding rewrite theory remains in its essence unchanged expect the types of the elements to be adapted accordingly.

7 Superposition of C/L laws on components

Superposition is the last step in our approach, in which a direct correspondence is established between the components and the different integrated compositional rules presented in the previous section. We will not consider situations where refinements of services are required at the component level. Besides that, the superposition itself consists just in adding the right

rule to the behaviour of the component concerned. In the near future, we are planning to demonstrate how such a superposed rule can easily be written in any concrete language, be it Java or XML, and thereby bring this work closer to service-oriented computing.

The process of superposition consists in checking that some instantiated event(s) matching an event in the integrated compositional rule has occurred in a given component. In this case, we have also to do the same checking of the context boolean expression from the running component to concerned compositional coordination/location rewrite rule.

By denoting concrete components for instance by $\{PAR@LOC_i\}$, we can capture the above required instantiations of events and parts of context through adequate *substitutions*. That is, we propose to use the following notation:

- $\sigma_{Cxtb}^{CP_{par@loc_i}} : X_{Cxtb} \rightarrow T_{Cxtb}(X)$ will correspond to the substitution related to the context. X_{Cxtb} in this case will contains, for instance, the location sorts and variables to be instantiated by concrete locations from respective components, whereas $T_{Cxtb}(X)$ will include boolean context operations like BT and $REACH$.
- $\sigma_{ev}^{CP_{par@loc_i}} : X_{evt} \rightarrow T_{Evnt}(X)$ denotes the substitution corresponding to the instantiation of events. In the same way X_{evt} will correspond to parameters variables, but also to location sorts to be instantiated by specific concrete locations from respective components.

Given this, the superposition inference rule may be formulated through the following definition:

Definition 7.1 (Superposition of integrated Compositional Cor-laws**- and** Loc-laws**-specification)** Given an integrated compositional Cor-laws- and Loc-laws-specification $(C_c@L_l, P_c@P_l, s_{c@l}, tr_{pc@pl}, tr_{c@l}, S_{Evnt_{c@l}}, tr_{ev})$, the superposition on running components we denote by $\{PAR@LOC_i\}$ is achieved through the following inference rule.

$\forall\, c_k \in C_c, \forall\, [n_{c_k}], [n'_{c_k}] \in [T_{S_C}(X) \cup T_{Msg_c}(X)]$, and $\forall\, p_{c_i} \in P_c, \forall\, [m_{p_{c_i}}], [m'_{p_{c_i}}] \in [T_{S_{P_c}}(X) \cup T_{Msg_{P_c}}(X)], \forall\, [ev_{p_{c_i}}] \in [T_{Evnt_{P_c}}(X)]$; $i \in \{1, .., |S_{P_c}|\}$.
$\forall\, l_o \in L_l, \forall\, [s_{l_o}], [s'_{l_o}] \in [T_{S_L}(X) \cup T_{Msg_l}(X)]$, and $\forall\, p_{l_j} \in P_l, \forall\, [r_{p_{l_j}}], [r'_{p_{l_j}}] \in [T_{S_{P_l}}(X) \cup T_{Msg_{P_l}}(X) \cup T_{Evnt_{P_l}}(X)]$; $j \in \{1, .., |S_{P_l}|\}$.

$$\exists\, \sigma_{ev}^{Par@Loc_j} : X_{evt} \rightarrow T_{Evnt}(X),\ /\ \overline{\sigma}_{ev}^{Par@Loc_j}([ev_{c@lc_j}]) \in ev^{Par@Loc_j}$$
$$\exists\, \sigma_{Cxtb}^{Par@Loc_j} : X_{Cxtb} \rightarrow T_{Cxtb}(X),\ /\ \overline{\sigma}_{Cxtb}^{Par@Loc_j}([Cxtb_{l_j}]) \in Cxtb^{Par@Loc_j}$$

$$\frac{\begin{array}{l} \forall rl_{c@l_k} \in \mathcal{R}_{\mathcal{C}@\mathcal{L}} \\ \quad rl_{c@l_k} :: Cxtb_{l_k} : \bigotimes_{j=1} (p_{c@lc_j}, [ev_{c@lc_j}]\ [m_{pc_j}]\ [r_{pl_j}]) \otimes (c@lc_k, [n_{c_k}]\ [s_{l_o}]) \\ \qquad \Rightarrow \bigotimes_{j=1} (p_{c@lc_j}, [m'_{pc_j}]\ [s'_{lc_o}]) \otimes (c@lc_k, [n'_{c_k}]\ [t'_{lc_o}])\ \mathtt{if}\ Ccnd \wedge Lcond \end{array}}{\begin{array}{l} \bigcup_{j=1,..,|S_L|} \{(Par@Loc_j,\ rl_{c@l_j} :: Cxtb_j : (p_{c@lc_j}, [ev_{c@lc_j}]\ [m_{pc_j}]\ [r_{pl_j}]) \otimes (c@lc_k, [n_{c_k}]\ [s_{l_o}]) \\ \qquad \Rightarrow (p_{c@lc_j}, [m'_{pc_j}]\ [s'_{lc_o}]) \otimes (c@lc_k, [n'_{c_k}]\ [t'_{lc_o}])\ \mathtt{if}\ Ccnd_j \wedge Lcond_j)\} \end{array}}$$

$\Diamond$

Informally speaking, when the associated part of context expression of the rule is an instance of the existing running context in some component (i.e. $\overline{\sigma}_{Cxtb}^{Par@Loc_j}([Cxtb_{l_j}]) \in Cxtb^{Par@Loc_j}$) and the same for the triggering events (i.e.$\overline{\sigma}_{ev}^{Par@Loc_j}([ev_{c@lc_j}]) \in ev^{Par@Loc_j}$) . In this case, we can dispatch the corresponding location/coordination rewrite rule to different components (i.e. as union of several rules, each in the associated components).

Remark 7.1 *For superposition to be correctly performed all these rules must be performed synchronously. Using the reflection capabilities of Rewriting Logic and the* MAUDE *language, this requirement can be directly fulfilled. However, we refrain from going into detail in this present report.*

Example 7.1 For our running example, the superposition concerns the integrated compositional rule we detailed in Example 6.1. As shown in Figure 3, the superposition pre-condition holds, that is: (1) there is a withdrawal event occurring from a specific customer (e.g. `cust1`); (3) he/she is the owner of the account; and (3) `BT` holds between the used atm and the corresponding bank. With this situation, the corresponding integrated rule is to superposed on the corresponding running components, namely `CUSTOMER@ATM` and `ACCOUNT@BANK`. As stated in the inference rule, this superposition consists in adequately splitting the rule parts among the components. That is, the parts that involve the Customer and/or the ATM (resp. Account and/or Bank) are to be superposed on the component `CUSTOMER@ATM` (resp. `ACCOUNT@BANK`). This superposition mechanism is illustrated in Figure 4. Notice finally that the part corresponding the located contract itself is to duplicated in both components (i.e. $(WdrsdP@ATMwdrP, \langle sdwdr|acco, cust\rangle\langle atmwdr|atm, bank\rangle))$.That is, this part allows informing both components that this specific customer is on standard withdrawal with its cooresponding account, and that its bank is this specific bank. More precisely, the instance cooreponding to that should also be brought from the coordination and location contracts related to this integrated law. ♢

Finally, it is worth mentioning that the similar superposition inference rule can be formulated allowing to superpose not only integrated rules but also independent compositional coordination and location rules. That is, without passing through the third step, we can superpose the compositional rules directly. This inference is given as follows.

Definition 7.2 (**Superposition Compositional** COR-LAWS**- and** LOC-LAWS**-specification)** Given two compositional COR-LAWS- and LOC-LAWS-specifications $(C_c, P_c, s_c, tr_{pc}, tr_c)$ and $(L_l, P_l, s_l, tr_{pl}, tr_l)$ the superposition on running components denoted by $\{PAR@LOC_i\}$ is achieved through the following inference rule.

$\forall\, c_k \in C_c, \forall\, [n_{c_k}], [n'_{c_k}] \in [T_{S_C}(X) \cup T_{Msg_c}(X)]$, and $\forall\, p_{c_i} \in P_c, \forall\, [m_{p_{c_i}}], [m'_{p_{c_i}}] \in [T_{S_{P_c}}(X) \cup T_{Msg_{P_c}}(X)], \forall\, [ev_{p_{c_i}}] \in [T_{Evnt_{P_c}}(X)]$; $i \in \{1, .., |S_{P_c}|\}$.
$\forall\, l_o \in L_l, \forall\, [s_{l_o}], [s'_{l_o}] \in [T_{S_L}(X) \cup T_{Msg_l}(X)]$, and $\forall\, p_{l_j} \in P_l, \forall\, [r_{p_{l_j}}], [r'_{p_{l_j}}] \in [T_{S_{P_l}}(X) \cup T_{Msg_{P_l}}(X) \cup T_{Evnt_{P_l}}(X)]$; $j \in \{1, .., |S_{P_l}|\}$.

$$\exists\, \sigma_{ev}^{Par@Loc_j} : X_{evt} \rightarrow T_{Evnt}(X),\ /\ \overline{\sigma}_{ev}^{Par@Loc_j}([ev_{c@lc_j}]) \in ev^{Par@Loc_j}$$
$$\exists\, \sigma_{Cxtb}^{Par@Loc_j} : X_{Cxt} \rightarrow T_{Cxtb}(X),\ /\ \overline{\sigma}_{Cxtb}^{Par@Loc_j}([Cxtb_{l_j}]) \in Cxtb^{Par@Loc_j}$$

$$\forall rl_{c_{i_1}} \in \mathcal{RC}, \forall rl_{l_{j_1}} \in \mathcal{RL}$$

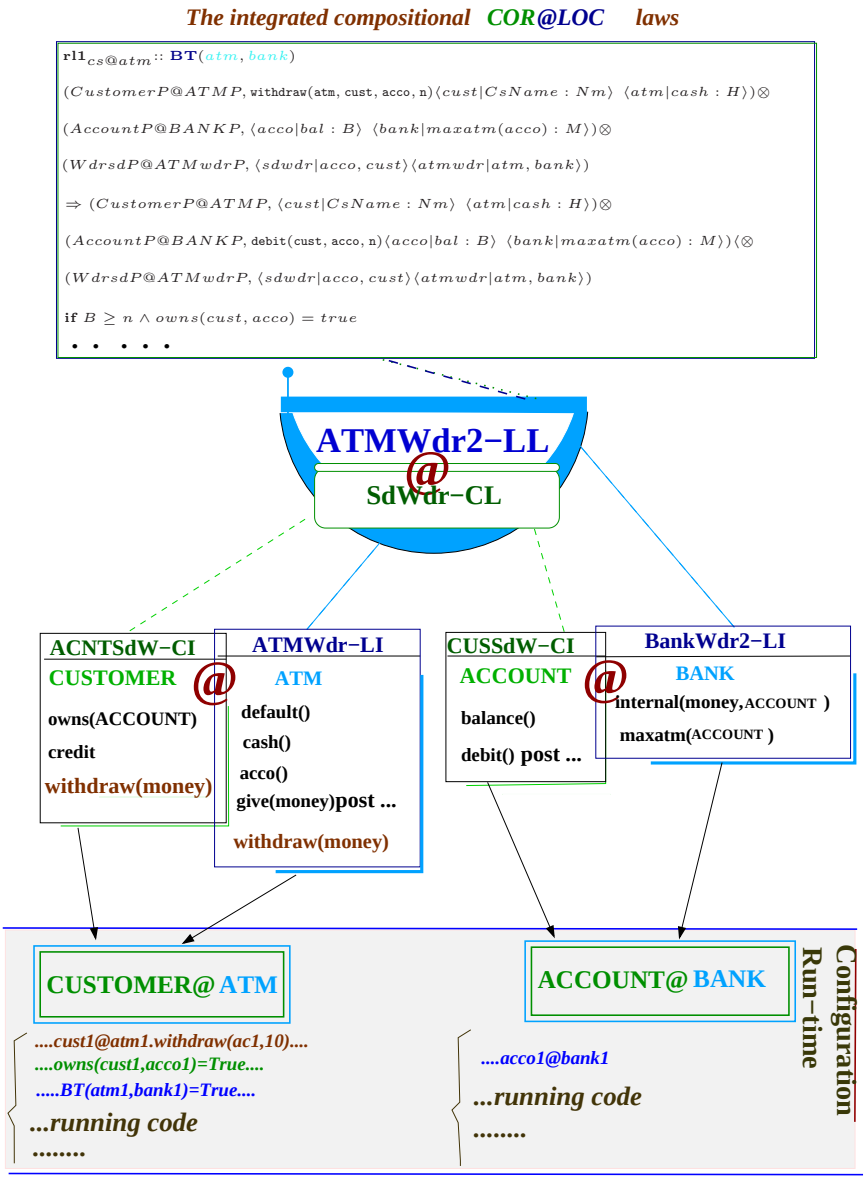


Figure 3: The Running Configuration State Before the Superposition.

$$rl_{c_{i_1}} : \underset{i=1}{\otimes}(p_{c_i}, [ev_{c_i}]\ [m_{p_{c_i}}]) \otimes (c_k, [n_{c_k}]) \Rightarrow \underset{i=1}{\otimes}(p_{c_i}, [m'_{p_{c_{i_1}}}]) \otimes (c_k, [n'_{c_k}])\ \texttt{if}\ Ccnd$$

$$rl_{c@l_{k_1}} :: Cxtb_{l_{k_1}} : \underset{j=1}{\otimes}(p_{c@lc_j}, [ev_{c@lc_j}]\ [m_{pc_j}]\ [r_{pl_j}]) \otimes (c@lc_k, [n_{c_k}]\ [s_{l_o}])\texttt{if}\ Lcnd$$

$$\bigcup_{j=1,..,|S_L|}\{(Par@Loc_j,\ rl_{c@l_j} :: Cxtb_j : (p_{c@lc_j}, [ev_{c@lc_j}]\ [m_{pc_j}]\ [r_{pl_j}]) \otimes (c@lc_k, [n_{c_k}]\ [s_{l_o}])$$

$$\Rightarrow (p_{c@lc_j}, [m'_{pc_j}]\ [s'_{lc_o}]) \otimes (c@lc_k, [n'_{c_k}]\ [t'_{lc_o}])\ \texttt{if}\ Ccnd_j \wedge Lcond_j)\}$$

$\Diamond$

We can also go one step further and directly superpose the coordination/location rules from the first step (i.e. without any validation or separately from the validation). To express the fact that these are not validated, we propose to represent coordination rules in RC instead as data structure of the form:

$\langle$ `rl-label| lefthand-side, righhand-side, condition`$\rangle$,

whereas location rules in RL will be of the form:

$\langle$ `rl-label, context| lefthand-side, righhand-side, condition`$\rangle$,

Following this representation, the superposition is achieved through the following inference rule:

Definition 7.3 (**Superposition** COR-LAWS**- and** LOC-LAWS**-specification)** Given two COR-LAWS- and LOC-LAWS-specifications $SP = \prec CTS, \mathcal{RC} \succ_{\mathcal{RT}_{\mathcal{C}}}$ and $SP = \prec LTS, \mathcal{RL} \succ_{\mathcal{RT}_{\mathcal{L}}}$, the superposition on running components we denote by $\{PAR@LOC_i\}$ is achieved through the following inference rule.

$\forall\, c_k \in C_c, \forall\, [n_{c_k}], [n'_{c_k}] \in [T_{S_C}(X) \cup T_{Msg_c}(X)]$, and $\forall\, p_{c_i} \in P_c, \forall\, [m_{p_{c_i}}], [m'_{p_{c_i}}] \in [T_{S_{P_c}}(X) \cup T_{Msg_{P_c}}(X)], \forall\, [ev_{p_{c_i}}] \in [T_{Evnt_{P_c}}(X)]$; $i \in \{1, .., |S_{P_c}|\}$.
$\forall\, l_o \in L_l, \forall\, [s_{l_o}], [s'_{l_o}] \in [T_{S_L}(X) \cup T_{Msg_l}(X)]$, and $\forall\, p_{l_j} \in P_l, \forall\, [r_{p_{l_j}}], [r'_{p_{l_j}}] \in [T_{S_{P_l}}(X) \cup T_{Msg_{P_l}}(X) \cup T_{Evnt_{P_l}}(X)]$; $j \in \{1, .., |S_{P_l}|\}$.

$$\exists\, \sigma_{ev}^{Par@Loc_j} : X_{evt} \rightarrow T_{Evnt}(X),\ /\ \overline{\sigma}_{ev}^{Par@Loc_j}([ev_{c@lc_j}]) \in ev^{Par@Loc_j}$$
$$\exists\, \sigma_{Cxtb}^{Par@Loc_j} : X_{Cxtb} \rightarrow T_{Cxtb}(X),\ /\ \overline{\sigma}_{Cxtb}^{Par@Loc_j}([Cxtb_{l_j}]) \in Cxtb^{Par@Loc_j}$$

$$\frac{\begin{array}{c}\langle rl_{c_{i_1}} \mid \underset{i=1}{\cdots}[ev_{c_i}]\ [m_{p_{c_i}}]\ [n_{c_k}]\ ,\ \underset{i=1}{\cdots}[m'_{p_{c_i}}]\ [n'_{c_k}]\ ,\ \texttt{C}cnd\rangle \in \mathcal{RC} \\ \langle rl_{l_{j_1}}\ , Cxtb_{j_1} \mid \underset{j=1}{\cdots}[ev_{l_j}]\ [r_{p_{l_j}}]\ [s_{l_o}]\ ,\ \underset{j=1}{\cdots}[r'_{p_{l_j}}]\ [s'_{l_o}]\ ,\ \texttt{L}cnd\rangle \in \mathcal{RL}\end{array}}{\begin{array}{c}\bigcup_{j=1,..,|S_L|}\{(Par@Loc_j,\ rl_{c@l_j} :: Cxtb_j : (p_{c@lc_j}, [ev_{c@lc_j}]\ [m_{pc_j}]\ [r_{pl_j}]) \otimes (c@lc_k, [n_{c_k}]\ [s_{l_o}]) \\ \Rightarrow (p_{c@lc_j}, [m'_{pc_j}]\ [s'_{lc_o}]) \otimes (c@lc_k, [n'_{c_k}]\ [t'_{lc_o}])\ \texttt{if}\ Ccnd_j \wedge Lcond_j)\}\end{array}}$$

$\Diamond$

8 Conclusions

We proposed a semantic framework based on Rewriting Logic for interpreting and validating an architectural approach to system construction and evolution that separates between Computation, Coordination and Distribution concerns. We showed how modelling primitives addressing these three dimensions can be expressed in this logic, allowing for compositional validation, superposition, and rapid prototyping by using the current implementation of MAUDE.

Context-awareness and mobility-related aspects are among the most topical research priorities in both academia and industry. Although it is too early for an exhaustive survey, we would like to mention related work that has inspired our approach and/or pointed to future developments. First, and to the best of our knowledge, there is no *conceptual modelling* approach that addresses location-awareness in business processes in the sense that we have motivated, except for the work in [Abo01], one of our main sources of inspiration. This work invokes the notion of "channel" for addressing location-awareness. It is, altogether, rather "operational", not as declarative as we wish ours to be, because it uses state machines as a modelling tool. It does not cope with the evolutionary side either, and it has not been integrated within an architectural approach that provides explicit connectors that can handle location-dependency aspects. Architectures for context-aware and mobile

systems have received considerable attention in the last few years [Dey00, Dey03, CEM03, MRM03, GSSS02, PPSGS04, RJP04] and influenced our own work around CommUnity [FLW03, LFW02, LF03]. However, the more business related aspects that one can find in the area of information systems has been largely ignored.

The work that we have exposed is being enriched in various ways. First, concrete implementation scenarios using the Maude language are being investigated. Second, as we pointed out in the paper, this work can lead to a framework suitable for specifying/validating applications within the service-oriented computing paradigm. Third, the present operational semantics needs to be enhanced in various ways. We plan to enrich it with more complete semantics using graph rewriting transformations, which have already proved to be suitable for mobile systems [GH04, FMT02], namely in the context of CommUnity [CH04, BFL+04]. Last but not least, we are considering introducing suitable temporal rewrite theories for analysis and verification purpose using the derived sequent proofs from each of the developed rewrite theories.

Acknowledgement

We would like to thank C.Oliveira with whom we developed Location Laws. We are also grateful to all our partners in AGILE for their comments and input.

References

[ABB+03] L. Andrade, P. Baldan, H. Baumeister, R. Bruni, A. Corradini, R. De Nicola, J. Fiadeiro, F. Gadducci, S. Gnesi, P. Hoffman, N. Koch, P. Kosiuczenko, A. Lapadula, D. Latella, A. Lopes, M. Loreti, M. Massink, F. Mazzanti, U. Montanari, C. Oliveira, R. Pugliese, A. Tarlecki, M. Wermelinger, M Wirsing, and A. Zawlocki. Agile: Software architecture for mobility. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *WADT*, volume 2755 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2003.

[Abo01] L. Abom. Frame-working RM-ODP in Banking. In A.J. Moinhos Cordeiro and H. Kilov, editors, *WOODPECKER 2001, In conjunction with ICEIS, 2001*. ICEIS Press, 2001.

[AF02] L. Andrade and J. Fiadeiro. Agility through coordination. *Information Systems*, 27:411–424, 2002.

[AF03] L. Andrade and J. Fiadeiro. Architecture Based Evolution of Software Systems. In M.Bernardo and P.Inverardi, editor, *Proc. of Formal Methods for Software Architectures*, volume 2804 of *Lecture Notes in Computer Science*, pages 148–181. Springer, 2003.

[AFO04a] N. Aoumeur, J. Fiadeiro, and C. Oliveira. Distribution Concerns in Service-Oriented Modelling. In S.Weerawarana, editor, *2nd International Conference on Service Oriented Computing (ICSOC'04), Short papers*, IBM REsearch Divison, IBM Report:RA221 (W0411-084), pages 26–35, 2004.

[AFO04b] N. Aoumeur, J. Fiadeiro, and C. Oliveira. Towards an Architectural Approach for Location-aware Business Processes. In *Proc. of the 13th IEEE International Workshops on Enabling, Technologies : Infrastructure for Collaborative Enterprises, June 14-16*, pages 147–152. IEEE Computer Society, 2004.

[AGKF02] L. Andrade, J. Gouveia, G. Koutsoukos, and J. Fiadeiro. Coordination Contracts, Evolution and Tools. *Software Maintenance and Evolution: Research and Practice*, 14(5):353–369, 2002.

[AS02] N. Aoumeur and G. Saake. A Component-Based Petri Net Model for Specifying and Validating Cooperative Information Systems. *Data and Knowledge Engineering*, 42(2):143–187, August 2002.

[AS04] N. Aoumeur and G. Saake. Dynamically Evolving Concurrent Information Systems :A Component-Based Petri Net Proposal. *Data and Knowledge Engineering*, 50(2):117–173, 2004.

[BFL+04] R. Bruni, J. Fiadeiro, I. Lanese, A. Lopes, and U. Montanari. New Insights on Architectural Connectors. In J. Lévy, E.W. Mayr, and J.C. Mitchell, editors, *IFIP TCS*, pages 367–380. Kluwer, 2004.

[BKKM02] P. Borovansky, C. Kirchner, H. Kirchner, and P. Moreau. Elan from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.

[CDE+99] M. Clavel, F. Duran, S. Eker, J. Meseguer, and M. Stehr. Maude : Specification and Programming in Rewriting Logic. Technical report, SRI, Computer Science Laboratory, March 1999. URL : http://maude.csl.sri.com.

[CEM03] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. Software Eng.*, 29(10):929–945, 2003.

[CH04] A. Corradini and D. Hirsch. An Operational Semantics of CommUnity based on Graph Transformation Systems. In R. Heckel, editor, *International Workshop on Graph Transformation and Visual Modeling Techniques. Satellite of ETAPS 2004*, volume 109 of *Electronic Notes in Theoretical Computer Science*, pages 111–124. Elsevier, 2004.

[CM96] M. Clavel and J. Meseguer. Reflection and Strategies in rewriting logic. In G. Kiczales, editor, *Proc. of Reflection'96*, pages 263–288. Xerox PARC, 1996.

[DELM00] F. Duran, S. Eker, P. Lincoln, and J Meseguer. Principles of Mobile Maude. In J. Meseguer, editor, *Proc. of the Second International Symposium on Agent Systems and Applications and Fourth on Mobile Agents (ASA/MA 2000)*, pages 73–85. Springer, 2000.

[Den98] G. Denker. From Rewrite Theories to Temporal Logic Theories. In H. Kirchner and C. Kirchner, editors, *Proc. of Second International Workshop on Rewriting Logic*, volume 15 of *Electronic Notes in Theoretical Computer Science*, 1998.

[Dey00] A.K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, 2000.

[Dey03] A.K. Dey. *Towards a Services Platform for Context-Aware Applications*. PhD thesis, 2003.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter 6, pages 243–320. North-Holland, Amsterdam, 1990.

[Fia04] J. Fiadeiro. *Categories for Software Engineering*. Springer, 2004.

[FL04] J. Fiadeiro and A. Lopes. CommUnity on the Move: Architectures for Distribution and Mobility. In *Formal Methods for Components and Objects*, volume 3188 of *Lecture Notes in Computer Science*, pages 177–198. Springer, 2004.

[FLW03] J. Fiadeiro, A. Lopes, and M. Wermelinger. A mathematical semantics for architectural connectors. In *Generic Programming*, volume 2793 of *Lecture Notes in Computer Science*, pages 190–234. Springer, 2003.

[FMOM+00] J. Fiadeiro, N. Marti-Oliet, T. Maibaum, J. Meseguer, and I. Pita. Towards a Verification Logic for Rewriting Logic. In D. Bert and C. Choppy, editors, *Recent Trends in Algebraic Development Techniques*, volume 1827 of *Lecture Notes in Computer Science*, pages 438–458. Springer, Berlin, 2000.

[FMT02] G.L. Ferrari, U. Montanari, and M. Tuosto. Graph-Based Models of Internetworking Systems. In *10th Anniversary Colloquium of UNU/IIST*, pages 242–266, 2002.

[GH04] P. Guo and R. Heckel. Modeling and Simulation of Context-Aware Mobile Systems. In *Proceeding of 19th IEEE Conference on Automated Software Engineeirng (ASE'04)*, pages 430–433. IEEE Computer Society, 2004.

[GKW+02] J. Gouveia, G. Koutsoukos, M. Wermelinger, L. Andrade, and J. Fiadeiro. The Coordination Development Environment. In R. Kutsche and H. Weber, editors, *FASE*, volume 2306 of *Lecture Notes in Computer Science*, pages 323–326. Springer, 2002.

[GSSS02] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing, special issue on "Integrated Pervasive Computing Environments"*, 21(2):22–31, 2002.

[GWM+92] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud. Introducing OBJ. Technical Report SRI-CSL-92-03, Computer Science Laboratory, SRI International, 1992.

[KL04] P. Kardasis and P. Loucopoulos. Expressing and Organising Business Rules. *Information and Software Technology*, 2004.

[Lec96] U. Lechner. Object Oriented Specification of Distributed Systems in the μ-Calculus and Maude. In J. Meseguer, editor, *Proc. of the First Inter. Workshop on Rewriting Logic*, volume 4. Electronic Notes in Theoretical Computer Science, 1996.

[LF03] A. Lopes and J. Fiadeiro. On how Distribution and Mobility interfere with Coordination. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques*, volume 2755 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2003.

[LFW02] A. Lopes, J.L. Fiadeiro, and M Wermelinger. Architectural primitives for distribution and mobility. In *Proc. of ACM SIGSOFT Symp. on Foundations of Software Eng.*, pages 41–50. ACM Press, 2002.

[Mes90] J. Meseguer. Rewriting Logic as a unified Model of Concurrency. In Baeten, J. C. M. and Klop, J. W., editor, *13th Proc. of CONCUR'90*, volume 458 of *Lectures Notes in Computer Science*, pages 384–400, 1990.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model for concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[Mes93a] J. Meseguer. A Logical Theory of Concurrent Objects and its Realization in the Maude Language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Object-Based Concurrency*, pages 314–390. The MIT Press, 1993.

[Mes93b] J. Meseguer. Solving the Inheritance Anomaly in Concurrent Object-Oriented Programming. In *ECOOP'93 - Object-Oriented Programming*, volume 707 of *Lecture Notes in Computer Science*, pages 220–246. Springer, 1993.

[Mes98] J. Meseguer. Research Directions in Rewriting Logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany*. Springer, 1998.

[MOM96] N. Marti-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In J. Meseguer, editor, *Proc. of First International Workshop on Rewriting Logic*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 189–224, 1996.

[MPS03] A. Maurino, B. Pernici, and F.A. Schreiber. Adaptive Channel Behavior in Financial Information Systems. In *Proceedings of Information and Collaboration Systems CAISE'03 Workshop (June 16-20)*, pages 77–89, 2003.

[MRM03] M. Mikic-Rakic and N. Medvidovic. Adaptable Architectural Middleware for Programming-in-the-Small-and-Many. In M. Endler and D. Schmidt, editors, *Proc. Middleware 2003*, volume 2672 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.

[MT99] J. Meseguer and C. Talcott. A Partial Order Event Model for Concurrent Objects. In *Proc. of CONCUR'99*, volume 1664 of *Lecture Notes in Computer Science*, pages 415–430. Springer, 1999.

[PPSGS04] V. Poladian, J. Pedro Sousa, D. Garlan, and M. Shaw. Dynamic Configuration of Resource-Aware Services. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*. ACM Press, 2004.

[RJP04] G.C. Roman, C. Julien, and J. Payton. A Formal Treatment of Context-Awareness. In M. Wermelinger and T. Margaria-Steffen, editors, *Fundamental Approaches to Software Engineering (FASE), Joint to ETAPS 2004*, volume 2984 of *Lecture Notes in Computer Science*, pages 12–36. Springer, 2004.

[vdBMR02] M. G. J. van den Brand, P.-E. Moreau, and C. Ringeissen. The ELAN environment: a rewriting logic environment based on ASF+SDF technology. In Mark G. J van den Brand and Ralf Lämmel, editors, *Proceedings of the 2st International Workshop on Language Descriptions, Tools and Applications*. Electronic Notes in Theoretical Computer Science, Vol. 64, Nr. 3, 2002.

[WKA+03] M. Wermelinger, G. Koutsoukos, R. Avillez, J. Gouveia, L. Andrade, and J. Fiadeiro. Using Coordination Contracts for Flexible Adaptation to Changing Business Rules. In *IWPSE*, pages 115–120. IEEE Computer Society, 2003.

[WKL03] W.M.N. Wan-Kadir and P. Loucopoulos. Relating Evolving Business Rules to Software Design. *Journal of Systems Architecture*, 2003.

[WKL+04] M. Wermelinger, G. Koutsoukos, H. Lourenco, R. Avillez, J. Gouveia, L. Andrade, and J. Fiadeiro. Enhancing Dependability Through Flexible Adaptation to Changing Requirements. In R. Lemos, C. Gacek, and A. Romanovsky, editors, *WADS*, volume 3069 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2004.

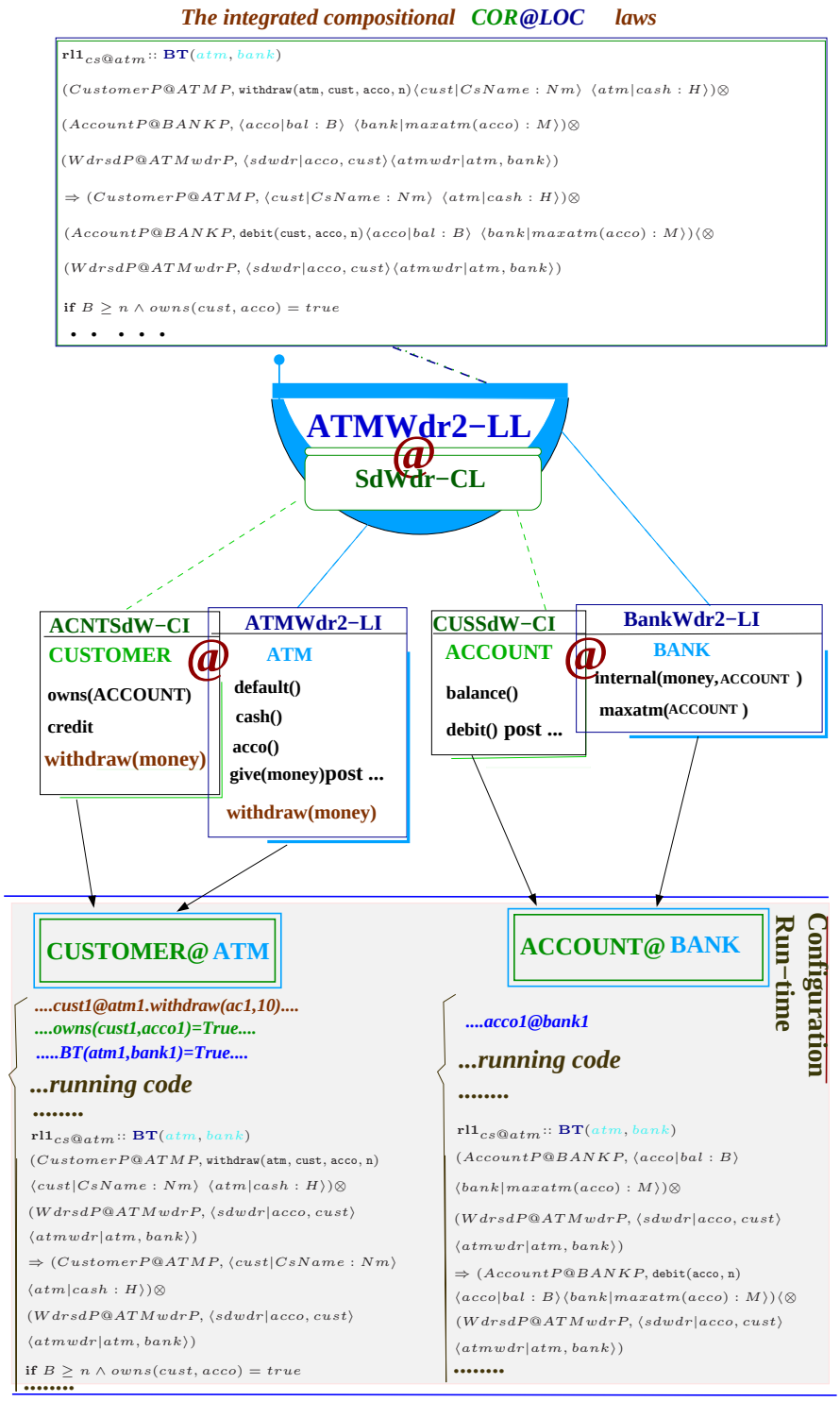


Figure 4: The Running Configuration State After the Superposition.