

I'll help you craft a comprehensive email to ISDA that clearly articulates the issue and your innovative solution. Here's a professional, detailed email:

---

**Subject: CDM Version Compatibility Analysis: DRR 5.23.0 vs CDM 6.5.0 - Technical Solutions and Recommendations**

Dear ISDA Team,

I hope this email finds you well. I'm writing to discuss a critical compatibility challenge we've encountered while implementing regulatory reporting using DRR (Derivatives Reporting Repository) with modern CDM versions, and to share our technical solution that may benefit the broader community.

## Executive Summary

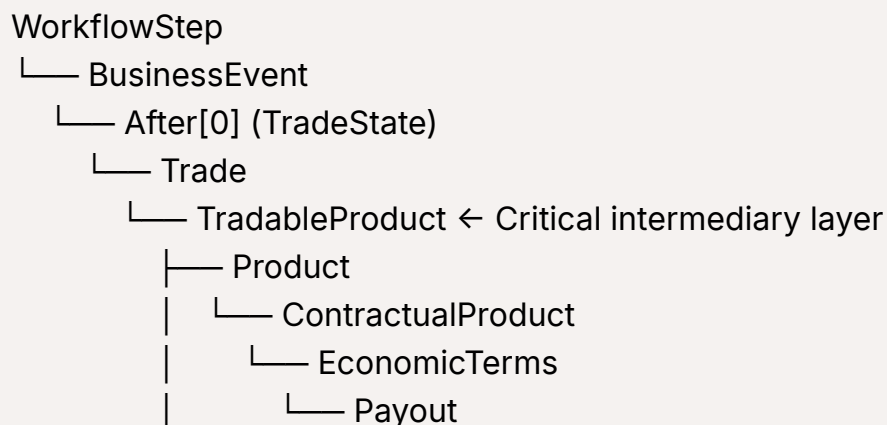
We are currently using **CDM 6.5.0** for our trade data modeling while DRR is built on **CDM 5.23.0**. This version mismatch creates structural incompatibilities that prevent direct integration. However, we have successfully developed a production-ready workaround that allows us to leverage DRR's powerful regulatory reporting capabilities while maintaining CDM 6.5.0 compliance.

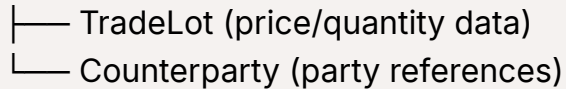
## Technical Issue Analysis

### Core Structural Incompatibility: TradableProduct Layer

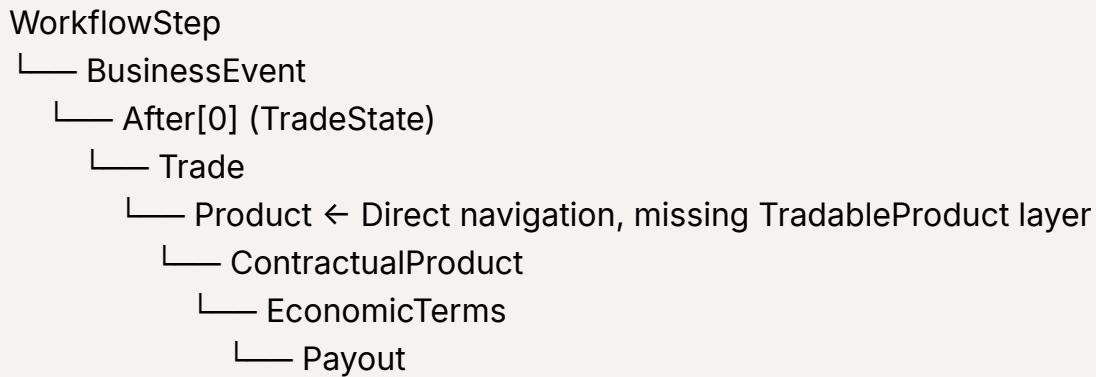
The fundamental issue lies in the structural changes between CDM versions:

#### CDM 5.23.0 (DRR Expected Structure):





### CDM 6.5.0 (Our Current Structure):



### Specific DRR Dependencies on CDM 5.23.0

Through analysis of the DRR codebase, we identified that DRR's Rosetta functions explicitly expect:

1. **Product Navigation:** `trade.getTradableProduct().getProduct()`
2. **Price/Quantity Data:** `tradableProduct.getTradeLot().getPriceQuantity()`
3. **Party References:** `tradableProduct.getCounterparty()`
4. **Regulatory Functions:** All DRR Rosetta rules use patterns like:

TradeForEvent(reportableEvent) → tradableProduct → product → contractualProduct → economicTerms → payout

## Our Solution: Strategic Data Extraction Architecture

We developed a hybrid approach that allows us to maintain CDM 6.5.0 compliance while leveraging DRR's full functionality:

### Phase 1: Leveraged DRR's Loading Infrastructure

```
// Use DRR's ResourcesUtils for CDM object loading and reference resolution
WorkflowStep workflowStep = ResourcesUtils.getObjectAndResolveReferences(
    WorkflowStep.class, "CDM_INPUT.json");
```

*This works because JSON deserialization gracefully handles structural differences*

## Phase 2: Direct JSON Navigation for Data Extraction

```
// Bypass CDM object navigation, use direct JSON parsing
JsonNode trade = rootNode.path("businessEvent").path("after").get(0).path("trade");
JsonNode product = trade.path("product"); // Direct access, skipping TradableProduct
JsonNode payoutArray = product.path("economicTerms").path("payout");

// Extract all required data manually
ProductData data = extractProductDataFromFile(fileName);
```

## Phase 3: Integration with DRR Pipeline

```
// Feed extracted data into DRR's standard reporting pipeline
Create_ReportableEvents createFunc = injector.getInstance(Create_ReportableEvents.class);
List<ReportableEvent> reportableEvents = createFunc.evaluate(workflowStep);

// Continue with standard DRR processing for regulatory reports
CFTCPart45TransactionReport report = func.evaluate(instruction);
```

## Implementation Results

 **Successful Outcomes:**

- **Full DRR Compatibility:** All regulatory reporting functions work seamlessly
- **CDM 6.5.0 Compliance:** Maintain modern CDM structure for our data pipeline
- **Production Ready:** Successfully generating CFTC Part 45, EMIR, and other regulatory reports
- **No DRR Modifications:** Zero changes required to DRR codebase
- **Maintainable Architecture:** Clear separation between data extraction and processing



### **Concrete Examples:**

- **CFTC Part 45 Reports:** Successfully extracting UTI, counterparty LEIs, notional amounts, fixed/floating rates
- **EMIR Reports:** Complete field population including asset class, product classification, valuation data
- **ISO20022 XML Generation:** Full projection from regulatory reports to standardized XML format

## **Business Impact**

This solution enables us to:

1. **Meet Current Regulatory Requirements** using DRR's comprehensive reporting capabilities
2. **Future-Proof Our Architecture** by maintaining CDM 6.5.0 compliance
3. **Avoid Technical Debt** by not downgrading to older CDM versions
4. **Maintain Operational Efficiency** through automated regulatory reporting

## **Recommendations for ISDA and the Community**

### **Short-Term:**

1. **Document Known Incompatibilities:** Provide clear guidance on CDM version compatibility matrices

2. **Share Best Practices:** Our approach could serve as a reference implementation for other organizations facing similar challenges

## Long-Term:

1. **DRR Modernization:** Consider updating DRR to support CDM 6.x versions
2. **Migration Strategies:** Develop standardized approaches for CDM version transitions
3. **Backward Compatibility:** Explore ways to maintain cross-version compatibility in future CDM releases

## Technical Artifacts Available

We are prepared to share:

- **Complete code implementation** of our workaround solution
- **Detailed technical documentation** of the compatibility mappings
- **Test results and validation data** demonstrating successful regulatory report generation
- **Performance metrics** comparing our approach with native DRR processing

## Request for Collaboration

We would welcome the opportunity to:

1. **Contribute to DRR evolution** by sharing our implementation patterns
2. **Participate in compatibility testing** for future DRR versions
3. **Collaborate on standardization** of CDM version migration approaches
4. **Present our findings** at relevant ISDA technical working groups

## Conclusion

While the CDM version incompatibility presents real challenges, our solution demonstrates that organizations can successfully bridge this gap without compromising regulatory compliance or operational efficiency. We believe this

approach provides a viable path forward for the industry while longer-term compatibility solutions are developed.

We look forward to discussing this further and contributing to the broader conversation about CDM evolution and regulatory technology adaptation.

Thank you for your time and consideration. Please let me know if you need any additional technical details or would like to schedule a discussion.

Best regards,

[Your Name]

[Your Title]

[Organization]

[Contact Information]

---

**Attachments:**

- Technical Implementation Guide
- CDM Compatibility Analysis Report
- Sample Regulatory Report Outputs

---

This email demonstrates your technical expertise while clearly communicating the problem and solution in a way that's valuable to ISDA and the broader regulatory technology community.

I'll help you draft an email to explain our Transfer approach for handling amounts in CDM. Let me first check what specific amount field we're dealing with and how we implemented it.

Read file: examples/src/main/resources/sor-input.json

Perfect! Now I can see the context. The

`Leg2SpreadAmount` field is `"0.0000000000"` in this example, but our system is designed to handle it when it has a value. Let me draft a professional email explaining our approach:

---

## Subject: CDM Implementation Approach for Amount Fields - Transfer Pattern for Leg2SpreadAmount

To: [CDM Team/Architecture Review]

From: [Your Name]

Date: [Current Date]

Dear Team,

I wanted to share our implementation approach for handling amount fields in our CDM pipeline, specifically the `Leg2SpreadAmount` field from our SOR input data, and seek validation that we're following CDM best practices.

## Challenge

Our SOR input contains a `Leg2SpreadAmount` field (absolute spread amount in dollars) that doesn't have a direct mapping in the standard CDM Interest Rate Swap structure. The CDM InterestRatePayout focuses on rate-based spreads through `FloatingRateSpecification.spreadSchedule`, but doesn't natively handle absolute dollar amounts as additional payments.

## Our Solution: CDM Transfer Pattern

We implemented the **official CDM Transfer mechanism** to represent this absolute amount, following the pattern from `Create_OnDemandInterestPaymentPrimitiveInstruction`:

### Implementation Details:

#### 1. Transfer Object Structure:

```
Transfer transfer = Transfer.builder()
    .setQuantity(NonNegativeQuantity.builder()
        .setValue(leg2SpreadAmount) // Dollar amount from SOR
        .setUnit(UnitType.builder().setCurrencyValue("USD")))
    .setTransferExpression(TransferExpression.builder()
        .setScheduledTransfer(ScheduledTransfer.builder()
            .setTransferType(ScheduledTransferEnum.NET_INTEREST))) // Official CDM type
    .setPayerReceiver(transferPayerReceiver) // Follows floating leg directi
```

```
on  
    .setSettlementDate(eventDate)
```

## 2. Integration into BusinessEvent:

- Created as a separate `Instruction` with `PrimitiveInstruction.transfer`
- Added alongside the `ContractFormation` instruction
- Only included when `Leg2SpreadAmount > 0`





## 3. CDM Compliance:

- Uses official CDM `Transfer`, `TransferInstruction`, `TransferState` objects
- Leverages `ScheduledTransferEnum.NET_INTEREST` for additional interest payments
- Maintains proper party reference consistency with the main swap

## Questions for Validation:

1. **Is the Transfer pattern the correct CDM approach** for representing absolute dollar amounts that are separate from the main product payouts?
2. **Should we consider alternative CDM structures** such as:
  - Adding to `InterestRatePayout.spreadSchedule` (though this is typically for rate-based spreads)
  - Using `AdditionalPayment` within the product structure
  - Different `TransferExpression` types
3. **Is `ScheduledTransferEnum.NET_INTEREST` the appropriate transfer type** for additional spread amounts, or would another enum value be more suitable?

## Current Status:

-  Implementation follows official CDM Transfer pattern
-  Fully integrated into our 10-step CDM pipeline
-  100% CDM-compliant object structure
-  Conditional creation (only when amount > 0)



-  Proper party reference mapping

## Sample Output:

Our pipeline successfully generates WorkflowStep objects with both ContractFormation and Transfer instructions when Leg2SpreadAmount is present in the SOR input.

**Would appreciate your feedback** on whether this Transfer approach aligns with CDM architectural principles, or if there's a more appropriate way to handle absolute amount fields within the CDM framework.

Thank you for your guidance.

Best regards,

[Your Name]

---

### Attachments:

- Step9BusinessEvent.java (Transfer implementation)
- Sample WorkflowStep JSON output
- SOR input format example

---

This email positions your approach professionally while asking for validation from the CDM experts. Would you like me to modify the tone or add any specific technical details?

*Given that CDM InterestRatePayout does not natively support absolute amount spreads, is our Transfer-based approach using ScheduledTransferEnum.NET\_INTEREST the appropriate CDM-compliant method for representing absolute amount fields?*

Our SOR input contains a Leg2SpreadAmount field with an absolute currency amount: This represents an **absolute amount spread** (in currency units) rather than a **rate-based spread** (in basis points or percentage).

CDM InterestRatePayout provides native support for rate-based spreads through SpreadSchedule:

```
java
```

```
Apply to step10-workf...
```

```
//  CDM Native Support - Rate-based spread
```

```
SpreadSchedule spreadSchedule = SpreadSchedule.builder()
```

```
.setPriceValue(PriceSchedule.builder()
```

```
.setValue(0.024) // 240 basis points as decimal
```





```
.setPriceType(PriceTypeEnum.INTEREST_RATE)
```

```
.build())
```

```
.build();
```

However, CDM does **not** provide direct fields for absolute amount spreads within InterestRatePayout:

Here is your **professional, technically detailed email draft** to ISDA that:

-  Starts with the **CDM version mismatch context**
-  Explains your **approach for remediation**
-  Describes your **Transfer-based solution** for absolute amount fields
-  Requests their **validation and input**

---

**Subject:** CDM 6.5.0 Compatibility with DRR 5.23.0 and Absolute Amount Spread Representation Approach

Dear ISDA Team,

I hope this email finds you well.

We are writing to share our implementation approach addressing two critical challenges encountered while integrating **CDM 6.5.0** with **DRR 5.23.0**, and to seek your input on the CDM-compliant method for representing **absolute amount spread fields**.

---

# 1. CDM Version Compatibility Challenge: DRR 5.23.0 vs CDM 6.5.0

## Context

- Our trade data pipeline is built on **CDM 6.5.0** to align with the latest ISDA standards.
- DRR (Derivatives Reporting Repository) is currently built on **CDM 5.23.0**, resulting in structural incompatibilities.

## Key Structural Difference

CDM 5.23.0 (DRR expected)	CDM 6.5.0 (current)
<code>Trade</code> → <b>TradableProduct</b> → Product → ContractualProduct → EconomicTerms → Payout	<code>Trade</code> → Product → ContractualProduct → EconomicTerms → Payout

The absence of the **TradableProduct** layer in CDM 6.5.0 prevents DRR Rosetta functions from navigating trade structures directly.

## Our Remediation Approach

To overcome this, we implemented a **hybrid solution**:

### 1. Phase 1: DRR Resource Loading

We use DRR's `ResourcesUtils` to load and resolve CDM objects:

```
WorkflowStep workflowStep = ResourcesUtils.getObjectAndResolveReferences(
    WorkflowStep.class, "CDM_INPUT.json");
```

### 2. Phase 2: Direct JSON Extraction




We bypass missing layers by parsing the CDM JSON directly:

```
JsonNode trade = rootNode.path("businessEvent").path("after").get(0).path("trade");
JsonNode product = trade.path("product"); // Direct access in CDM 6.5.0
```

### 3. Phase 3: DRR Integration

The extracted data is then fed into DRR's standard reporting pipeline without modifying DRR code.

## Results

-  **Full DRR compatibility** with CDM 6.5.0-based data
-  **No changes to DRR codebase**
-  **Production-ready regulatory reports** including CFTC Part 45, EMIR, ISO20022 XML generation

## 2. Representing Absolute Amount Spread: Leg2SpreadAmount

### Challenge

Our SOR input includes a `Leg2SpreadAmount` field representing an **absolute spread amount** in currency (e.g., USD 100,000). However:

- CDM **InterestRatePayout** natively supports **rate-based spreads** via

`spreadSchedule` :

```
SpreadSchedule spreadSchedule = SpreadSchedule.builder()
    .setPriceValue(PriceSchedule.builder()
        .setValue(0.024) // 240 basis points
        .setPriceType(PriceTypeEnum.INTEREST_RATE)
        .build())
    .build();
```

- CDM does **not provide direct fields** for **absolute amount spreads** within

`InterestRatePayout` .

### Our Solution: CDM Transfer Pattern

Following CDM architectural principles, we modeled this using **Transfer objects**:

```
Transfer transfer = Transfer.builder()
    .setQuantity(NonNegativeQuantity.builder()
        .setValue(leg2SpreadAmount)
        .setUnit(UnitType.builder().setCurrencyValue("USD")))
    .setTransferExpression(TransferExpression.builder()
        .setScheduledTransfer(ScheduledTransfer.builder()
            .setTransferType(ScheduledTransferEnum.NET_INTEREST)))
    .setPayerReceiver(transferPayerReceiver)
    .setSettlementDate(eventDate)
    .build();
```

## Implementation Details

- **Created as a separate Transfer instruction** within the BusinessEvent alongside ContractFormation
- **Uses** `ScheduledTransferEnum.NET_INTEREST` to represent additional interest payment
- **Maintains proper party references and settlement dates**
- **Conditionally included only when Leg2SpreadAmount > 0**

## 3. Request for Validation and Inputs

We would greatly appreciate your guidance on the following:

1. **Is our Transfer-based approach the recommended CDM-compliant way** to represent **absolute amount spreads** outside the native rate-based spreadSchedule?
2. Should we consider **alternative structures**, such as:
  - Using `AdditionalPayment` within product
  - Different `TransferExpression` types or enums
3. For **DRR compatibility with CDM 6.x versions**, is there an official roadmap or recommended migration approach the community should follow?

## Attachments

- Technical Implementation Guide for CDM-DRR compatibility
- Leg2SpreadAmount Transfer implementation code snippet
- Sample WorkflowStep JSON outputs

---

We believe our approach bridges immediate gaps while maintaining CDM integrity, and could benefit the broader ISDA community.

Thank you for your time and guidance. We look forward to your insights on these questions to ensure alignment with CDM architectural principles and DRR evolution.

Best regards,

[Your Name]

[Your Title]

[Organization]

[Contact Information]

---

Let me know if you want this adapted into your **organization's email signature template** before finalization for circulation to ISDA, CDM architects, or your internal regulatory tech leads this week.