# Compositional contract specification for REA (Abstract)

Fritz Henglein Ken Friis Larsen Jakob Grue Simonsen
Christian Stefansen
Department of Computer Science, University of Copenhagen (DIKU)
{henglein,kflarsen,simonsen,cstef}@diku.dk

September 19th, 2007

**Contracts** When entrepreneurs enter contractual relationships with a large number of other parties, each with possible variations on standard contracts, they are confronted with the interconnected problems of *specifying* contracts, *monitoring* their execution for performance[1], *analyzing* their ramifications for planning, pricing and other purposes prior to and during execution, and *integrating* this information with accounting, workflow management, supply chain management, production planning, tax reporting, decision support *etc.*

Andersen, Elsborg, Henglein, Simonsen and Stefansen [AEH+06] define a typed language for compositional contract specification:[2]

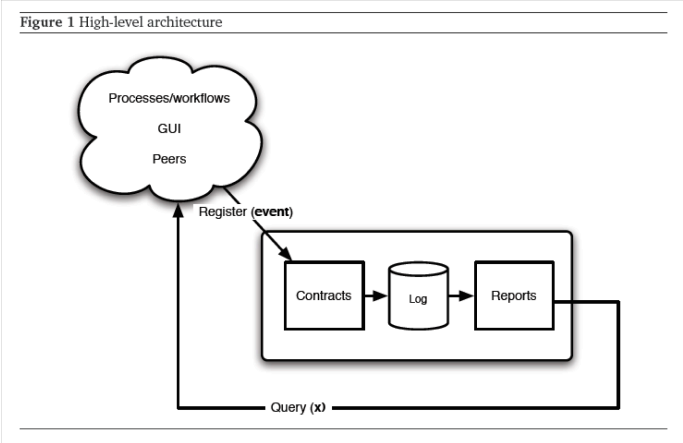
$$\begin{array}{rcl} c & ::= & \text{Success} \mid \text{Failure} \mid f(\vec{a}) \mid \text{transmit}(A_1, A_2, R, T \mid P).\, c \\ & \mid & c_1 + c_2 \mid c_1 \parallel c_2 \mid c_1; c_2 \end{array}$$

Success denotes the *trivial* or *(successfully) completed* contract: it carries no obligations on anybody. Failure denotes the *inconsistent* or *failed* contract; it signifies breach of contract or a contract that is impossible to fulfill. For a Boolean predicate $P$ the contract expression $\text{transmit}(A_1, A_2, R, T \mid P).\, c$ represents a contract where the *commitment* $\text{transmit}(A_1, A_2, R, T \mid P)$ must be satisfied first. The commitment must be *matched* by a *transmit event* $e = \text{transmit}(v_1, v_2, r, t)$ of resource $r$ from agent $v_1$ to agent $v_2$ at time $t$ such that $P(v_1, v_2, r, t)$ holds. After matching, the residual contract is $c$ in which $A_1, A_2, R, T$ are bound to $v_1, v_2, r, t$, respectively. Note that $A_1, A_2, R, T$ are binding variable occurrences whose scope is $P$ and $c$. In this fashion the subsequent contractual obligations expressed by $c$ may depend on the actual values in event $e$; such as a payment being due 8 days after delivery of the goods. The *contract combinators* $\cdot + \cdot, \cdot \parallel \cdot$ and $\cdot; \cdot$ compose subcontracts according to contract composition patterns: by alternation, concurrently, and sequentially, respectively. A (contract) context is a finite set of named contract template declarations of the form $f(\vec{X}) = c$. By using the *contract instantiation* (or *contract application*) construct $f(\vec{a})$ contract templates may be (mutually) recursive, which, in particular, captures repetition of subcontracts. Contract template definitions occur only at top level.

The language operates at two levels: the base level of (primitive commitments requiring the occurrence of) economic events such as transfer of resources between economic agents, reflecting the basic ontological concepts of the REA accounting

[1] *Performance* in contract lingo refers to *compliance* with the *promises* (contractual commitments) stipulated in a contract; nonperformance is also termed *breach of contract.*

[2] The types are elided here.




model [McC82]; and the compositional level of contract combinators. In commercial contracts only transfers of resources (goods, service, money) are included. Production events could be included, too, however. Indeed, with the compositional level being parametric in the base language, any kind of event types could be included in the base language, also outside the realm of economic events.

**A contract-based event-driven architecture** An event-driven architecture (EDA) [EDA06], is, loosely speaking, a software architecture that is organized around (data representing) *events* that *drive* system/component state transitions which in turn may generate events and other observable outputs.

We are presently working on developing a *contract-based* EDA for enterprise resource planning (ERP) systems [Wik]. Its high-level architecture is depicted in Figure 1. In this architecture, contracts such as standard or customized sales agreements, leases, *etc.* can be installed (entered) dynamically. Installed contracts are then matched against incoming events; after matching an event a contract is converted into an explicit representation—again as a contract—of the residual obligations.

Being *data* (residual) contracts have additional uses beyond monitoring their execution. They can be inspected, audited, analyzed and changed in response to failures to perform. Standard or customized report functions can be installed that at any point in time can be applied to the log of registered events alone (*ex-post* reports such as payments received) or, more interestingly, to *both* the log *and* the current contract states. An example of such an *ex-ante* analysis could be inventory restocking required to fulfil future demand based on both currently open orders and previously expedited orders. A basic ex-ante analysis for extracting deadline-ordered task lists has been described for the commercial contracts of Andersen *et al* [AEH$^+$06]. Peyton-Jones and Eber [JE03] have demonstrated sophisticated compositional pricing analysis for financial contracts. The key point here is that such analyses are defined once and for all for *all* definable contracts in an expressive language, not just a fixed finite set of *given* contract templates. Consequently, a custom contract not used before is automatically covered and does not require development of specialized analysis software.

Contracts can be thought of as declarative formal (behavioral, temporal) interface specifications in the spirit of software design by contract [Mey97], without any requirement for modeling real-world contracts. As such a contract functions as a behaviorial type for the process that generates the (expected) events. In ERP systems most basic processes that generate events such as order entry and financial bookkeeping are not automated, but performed by humans. Since a contract is an explicit representation of what events are expected (allowed) to happen next, contracts specifications can be used to automatically derive a user interface that prompts and guides the user through contract execution, guaranteeing that all and only relevant user interface options are provided at any given point during execution.

For an automated (executable) process that generates events contract residuation provides run-time verification. Conceivably, with both process code and contract specification in hand it should principally be possible to prove *statically* that a process always complies with its contract. This is bound to require a rather drastic limitation of expressive power of the base language to achieve practical analyzability while retaining sufficient expressiveness and generality for the intended domain-specific applications, however. Where such a "soft spot" is—and whether it exists at all—remains to be seen for now.

**Acknowledgements** The above reflects ongoing work within the 3d generation Enterprise Resource Planning Systems Project (3gERP.org), a collaboration between Copenhagen Business School, University of Copenhagen and Microsoft Development Center Copenhagen made possible by a grant by the Danish National Advanced Technology Foundation.

The section on contracts is excerpted from Andersen, Elsborg, Henglein, Jakobsen, Stefansen [AEH+06]; Figure  is from Larsen, Simonsen, Stefansen [LSS07]; both with permission by the authors.

# References

[AEH+06] Jesper Andersen, Ebbe Elsborg, Fritz Henglein, Jakob Grue Simonsen, and Christian Stefansen. Compositional specification of commercial contracts. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):485–516, November 2006.

[EDA06] *Workshop on Event Driven Architecture*, October 2006. http://www.haifa.il.ibm.com/Workshops/oopsla2006/present.html.

[JE03] Simon Peyton Jones and Jean-Marc Eber. *How to Write a Financial Contract*. Palgrave Macmillan, 2003. In: The Fun of Programming.

[LSS07] Ken Friis Larsen, Jakob Grue Simonsen, and Christian Stefansen. Towards a new high-level architecture for ERP systems (position paper). October 2007. http://www.3gERP.org/workshop.

[McC82] William E. McCarthy. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review*, LVII(3):554–578, July 1982.

[Mey97] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997. ISBN 0-13-629155-4.

[Wik] Wikipedia. Enterprise resource planning. http://en.wikipedia.org.