A Work Allocation Language with Soft Constraints

Christian Stefansen[1] Sriram Rajamani[1] Parameswaran Seshan[2]

[1] Microsoft Research Lab India, Bangalore, India
{cstef@diku.dk|sriram@microsoft.com}
[2] SETLabs, Infosys Technologies Limited, Bangalore, India
parameswaran_seshan@infosys.com

Abstract. While today's workflow languages have sophisticated constructs for specifying flow of control and data, facilities for associating tasks in a workflow with humans are largely missing. This paper presents SoftAlloc, a workflow allocation language with *soft constraints*, and explains the requirements that lead to its design—in particular what soft constraints are and how they enable workflows to capture best practices and organizational goals without rendering the workflows too strict. SoftAlloc is parameterized over external data access functions and takes a user-database query sublanguage as a parameter, thus allowing SoftAlloc to be used with virtually any process language (such as *WS-BPEL* or the π-calculus based language *SMAWL* [1]).

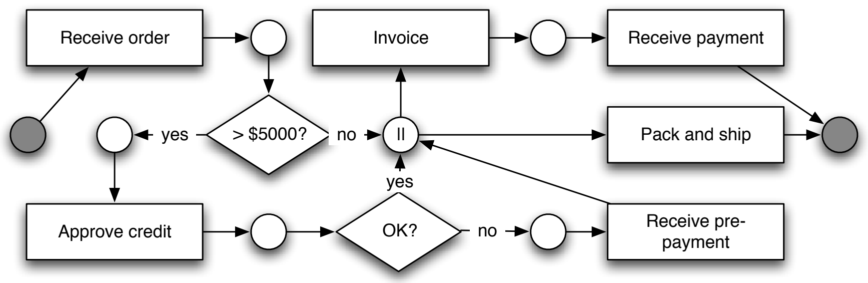
1 Introduction

Computer-orchestrated business processes are increasingly playing a direct role in how companies organize work. Computer-orchestrated processes now commonly involve both human resources and computer resources. Tasks can often be handled by many different resources. Therefore the process orchestration engine must decide in negotiation with the human resources who of the eligible resources ultimately carries out the task. Assigning a task to a resource is referred to as *allocation*. Allocating tasks to humans is inherently more complex than allocating to computer resources: in addition to having multiple, changing attributes that decide what they can, may or should do, humans have personal preferences and may choose to override the allocation rules at runtime, e.g. because they possess domain knowledge not captured in the system or because they make conscious, reflected violations to speed up processing in cases where where the process description focuses too narrowly on perfect compliance. For a treatise of these issues and *soft constraints* see Stefansen and Borch [2].

Consider the example workflow given in Fig. 1. In this workflow we might wish to say that the task *Pack and ship* should be carried out by someone with the role *Packing*, i.e. someone who is assigned to the packing unit. We can imagine specifying this by attaching a rule to the task saying:

```
role = "Packing"
```

Fig. 1 Order process (Petri net-style notation; || is a shorthand for parallel split)



Similarly, we may wish to assign the task *Receive order* to a resource with the role *Sales* who resides in the same locale as what is registered on the order. Additionally, we accept anyone with the role "Global sales". We then write:

```
    role = "Sales" and location = process.order.location
 or role = "Global sales"
```

Notice that we compose small building blocks of rules into larger rules. Now imagine that we want the steps *Invoice* and *Receive payment* to be carried out by the same person to retain familiarity. We would then attach a rule saying `role = "Finance"` to the task *Invoice* and then add

```
    role = "Finance"
and user = #whoDid ("Invoice", process.id)
```

as a rule to *Receive payment*.

But something is awry here. While we would certainly *prefer* the task *Receive payment* to be done by the same person who did the invoicing, this is only a *preference*—certainly not a strict rule that should be allowed to stand in the way of timely workflow completion if the designated person happens to be busy or temporarily absent. We have just committed one of the most common mistakes in workflow specification: we have promoted a *soft goal* to a strict rule and thereby created an inflexible system!

Alternatively, we might have removed the rule and only have said `role = "Finance"`, but that would have left out useful intentional information about our best practice. So just specifying fewer constraints is not attractive either.

This example illustrates that allocation constraints can represent a wide spectrum of specifications: some rules are best kept strict (e.g. *Expense approval* must be done by a *Manager*) while other rules are simply guidelines (e.g. *Replenish printer cartridges* should be allocated on a rotation basis (*round robin*)). The latter allocation strategy represents an organizational *soft goal*, which might have been "rotate tedious tasks between qualified workers to achieve a sense of fairness and variation and keep workers happy". This is undeniably a laudable goal, but if the company is experiencing peak load, this goal must temporarily yield to more mission-critical business goals (e.g. response time *vis-à-vis* our customers). Therefore, it cannot be written as a hard constraint, but leaving it out entirely renders the system unable to suggest the preferred person.

Going back to our example what we probably mean could be written as

```
       role = "Finance"
prefer [10] user = #whoDid("Invoice", process.id)
```

which states that we require a finance person to handle the activity under all circumstances, but we prefer the person who did the invoicing in that process. The number 10 represents a number of *points* to indicate how strong a preference this is. This becomes more interesting, when more preferences are in play. Consider the following rule for allocating the *Credit approval* step in the workflow:

```
       role = "Manager" or role = "Finance"
prefer [10] role = "Manager"
       [-#queueItems(user)]
```

The rule states that either *Manager* or *Finance* should handle the *Credit approval* task. A manager is preferred, but the number of items in the manager's queue is deducted from the preference level; i.e. someone with a short queue is preferred. Indeed, if all managers have more than 10 items in their queues, someone from *Finance* will be preferred in the interest of time. This shows how soft constraints in conjunction with hard constraints can be used to express soft goals and performance heuristics. Other typical soft constraints that we can then model are *round robin*, *prefer least loaded resource*, and *prefer shortest queue*.

Conceptually we can think of soft constraints being attached not just to a tasks, but also to a scope, a set of activities, a process or a resource itself—or they can be inserted on several levels to compose generic rules and policies with more specific ones. Soft constraints can also be used to express overall policies in the workflow engine. In this way the allocation rule language is really a hierarchical *scheduler programming language*.

2 Evaluation/Experience

The prototype has been tested and preliminarily evaluated with Infosys' PEAS platform and the language is slated for inclusion in the PEAS platform. Several prototype workflows have been tested with the allocation language, including a CRM (Customer Relations Management) workflow, a sales process, a approval/review process, and a bank transaction process.

Discussions have been held with domain experts from various industries, such as transportation, finance, insurance, call centers, etc. [3] The language overlaps with the delegating responsibilities of a call center floor manager. It would be a valuable tool to support floor managers in their decisions; if not render them entirely superfluous just yet. The insurance industry (in particular, claims processing) in many cases already has tailored systems, but those typically have a subset of the functionality here, and our language will become increasingly relevant as companies change to SOA. The transportation industry has different demands, in particular relating to scheduling with time and location, which is future work to add to SoftAlloc. The financial sector focuses mostly on transactions so the language is sufficient, if not slightly more than what is needed.

A patterns-based evaluation in the style of Russell *et al.* [4] was considered, but found unsuitable. It is essential to note that only the patterns that specify who is *ultimately* allowed to perform a task are included; our language did not set out to specify runtime negotiation protocols, which we consider an orthogonal concern and have handled more elegantly elsewhere in the architecture. While the language does indeed cover all the patterns intended, a patterns-based analysis inadequately captures the expressive power of our language. E.g. soft constraints represent an idea that could easily be expanded to comprise an entire suite of patterns in its own right, but the patterns-based analysis did not anticipate soft constraints and therefore does not cover them. Similarly, the patterns work does not mention if patterns can be combined and if so with what constraints.

3 Conclusion and Future Work

The language SoftAlloc can express all patterns for which it was designed and all examples that were deemed necessary. The use of soft constraints has proven extremely beneficial, and as intended the language integrates with any system we have seen. A GUI for the language is being developed in the production setting where the language is to be used. Based on the discussion of resource patterns, it would be interesting to construct a collection of soft constraint patterns.

Some benefits are yet to be reaped: the language is a small non-recursive DSL and this means that allocation rules are not only expressions that can be evaluated, but also pieces of data that can be used to perform static analysis. Performance simulation can be used to identify bottlenecks, estimate capacity requirements, and suggest what resources to add. This is an important improvement over previous systems, where the lack of integration made performance analysis a non-routine job requiring specialized skills.

References

1. Stefansen, C.: SMAWL: A SMAll Workflow Language based on CCS. Technical Report TR-06-05, Harvard University, Div. of Eng. and App. Sci. (March 2005) `http://stefansen.dk/papers/ccs-petrinet.pdf`.
2. Stefansen, C., Borch, S.E.: Using soft constraints to guide users in flexible business process management systems (BPMS). International Journal of Business Process Integration and Management **3**(1) (2008)
3. Stefansen, C., Rajamani, S., Seshan, P.: A work allocation language with soft constraints. Technical report, Microsoft Research India (2007) `http://stefansen.dk/papers/SoftConstraintAllocation.pdf`.
4. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow resource patterns. Technical report, Eindhoven University of Technology (2005)