Inference and propagation of performance constraints from abstract to concrete business workflows

A. García-Domínguez[a], I. Medina-Bulo[a], Mariano Marcos-Bárcena[b]

[a] *Department of Computer Languages and Systems, University of Cádiz, C/Chile 1, CP 11003, Cádiz. E-mail: {antonio.garciadominguez,inmaculada.medina}@uca.es*
[b] *Department of Mechanical Engineering and Industrial Design, University of Cádiz, C/Chile 1, CP 11003 Cádiz. E-mail: mariano.marcos@uca.es*

Abstract

Currently, manufacturing organizations of all sizes need to collaborate with others in order to meet their goals. However, achieving an adequate level of communication between their people and their information systems is difficult. Organizations are urged to formalize their processes and implement their information systems flexibly. These tasks can be achieved using workflows and service-oriented architectures, respectively. Abstract business-level workflows can be refined into executable workflows which compose services from the established service-oriented architecture, integrating disparate systems. However, it is hard to ensure that the target performance of the business process is achieved from the concrete software components. In this work, we present two algorithms which infer performance constraints for tasks in a workflow from its global and local restrictions. These tasks can be then further decomposed into nested subgraphs automatically. By repeatedly decomposing the tasks and inferring their constraints, target key performance indicator values can propagate from abstract business processes down to concrete activities.

Keywords: workflow modelling, service-oriented architectures, holonic enterprise, business processes, Web Services.

1. Introduction

In the current rapidly changing market, manufacturing enterprises need to focus on their key value adding processes in order to remain competitive. Non-core processes are outsourced or shared with external organizations. This is the case for both large firms and their collaborative manufacturing meganetworks (CMMNs) [1] and small firms which join together to compete with them [2]. Virtual enterprises pool the firms' resources into specific projects, and extended enterprises take form from trust relationships between the organizations [3].

However, these collaborations introduce new challenges. Business practices and software platforms will usually differ, hampering communications at the business and information system levels.

In order to reach a common understanding, enterprises need to agree on well-defined business processes in order to collaborate. At the same time, the information systems needs to implement these business processes, accommodating their reconfiguration and monitorization and integrating disparate software platforms.

In the business process modelling field, several workflow modelling languages are being developed,

such as BPMN 2.0 [4] or WS-BPEL 2.0 [5]. These workflows can describe the business process, run it and monitor its progress, seamlessly aggregating human and computer resources. Workflows can be used at every abstraction level, from the high-level business activities down to the individual activities performed by each participant. At the information system level, the basic building block in these workflows is a *service*: a self-contained piece of software which performs a task and can be reused across the entire organization.

However, the more detailed the model, the harder it is to see the high-level picture and check if the individual activities are meeting the high-level key performance indicators. When the desired level of performance is not obtained, it is important to be able to pinpoint the cause quickly. This could be achieved if the low-level activities were annotated with constraints which could be traced back to the high-level key performance indicators.

In this work, we propose a top-down approach to derive the desired performance of the individual activities performed by each resource from the target values of the key performance indicators of the abstract business process. After introducing some basic concepts and our simplified workflow modelling language, we present two algorithms which infer local performance constraints from global ones, and a transformation which expands the graph so the user can increase the level of detail. Finally, we offer some conclusions and future lines of work.

2. Background

Before presenting the algorithms, we will introduce some basic concepts and related works.

2.1. Workflow modelling

Formalizing the business practices of an organization is a common practice nowadays. A formal business process can guide the collaboration between different organizations or different parts of an organization, and provides a holistic view of all the work required. Aguilar-Savén classifies business process models into 4 categories, depending on their purpose [6]:

a) Learning about the existing process;
b) Improving or developing new processes;
c) Providing decision support during process execution and control; and
d) Supporting the enactment of Information Technology.

In this work, we are most interested in purposes b) and c): we want to model the existing or projected business process and transform it so the information systems can support their execution and control. There are quite a few alternatives for modelling business processes, such as function diagrams (i.e. IDEF0 [7]) or Gantt charts, but not all cover both purposes.

Workflows are defined by the Workflow Management Coalition as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". As such, they are clearly oriented towards purpose c), and can serve purpose b) depending on their abstraction level. Normally, these workflows are described in a specific language, supported by a Workflow Management System (WFMS). The WFMS communicates with the participants (both humans and software entities) to execute the workflows and monitor them.

There are several workflow modelling languages currently in use. Some of them are designed to be directly run and monitored by a WFMS automatically, such as the Web Service Business Process Execution Language (WS-BPEL) 2.0 [5]. Others work at the business level, describing the process in a more abstract way, such as the Business Process Modeling Notation (BPMN) 2.0 [4]. Usually, modellers start with abstract workflows such as BPMN, and later on derive concrete workflows (i.e. in WS-BPEL 2.0) from them.

2.2. Service-oriented architectures

Previously, we described how workflows can help an organization formalize their practices, assisting the improvement and control of their business processes. However, the information systems need to provide the basic building blocks to make them work.

This requires a change in the way the systems are conceived: instead of closed silos of information which are solely manipulated through a user interface, they must now provide a catalogue of *services* which can be flexibly reused over all workflows in the organization. These new systems are known as *service-oriented architectures* (SOAs).

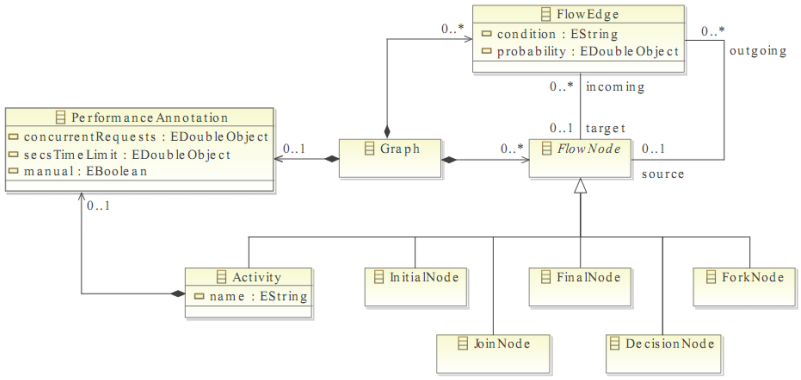

Figure 1. UML class diagram for the elements of our simplified workflow language

According to Erl, services "exist as physically independent software programs with distinct design characteristics that support the attainment of the strategic goals associated with service-oriented computing" [8]. The definition is intentionally vague: any combination of technologies can be used to build a SOA. However, in practice, most SOA development uses Web Services (WS) as defined by the W3C [9]. By using a standardized technology stack, users can interoperate between differing software platforms.

It is interesting to note the interactions between workflows as business processes and SOA: a business process can be modelled as a concrete WS-BPEL 2.0 composition of several existing Web Services. After performing this composition, the WS-BPEL 2.0 workflow is now yet another Web Service which can be reused from anywhere else in the organization. This allows for building more advanced business processes in a bottom-up fashion, and improve the processes more quickly. In a way, this view of the system as a network of collaborating entities which join simpler entities and take part in several others can be seen as a software implementation of the *holonic enterprise* [10].

At the same time, reusing the services to this degree increases our reliance in them: we depend on their performance in order to meet the target values for the key performance indicators of the business processes. However, it is hard to know if we are meeting these high-level indicators from the individual services. For this reason, we propose in this paper several techniques to derive the required performance values for each service, from the high-level abstract workflows down to the low-level, executable workflows.

3. Inference of performance constraints

In the previous sections, we introduced the key concepts required to understand the work presented in this paper, and our motivation. In this section, we will describe the inference algorithms used to annotate the workflow activities with the expected performance, using the global and local information set by the modeller on the workflow.

3.1. Workflow language

We use a simplification of UML activity diagrams to describe our workflows [11]. Figure 1 shows an UML diagram class with each of the element types:

- **Activities** encapsulate some behaviour, described textually.
- Activities can have manual or automatic **performance annotations** on the number of requests they must handle per second and the time limit for each of these requests.
- **Initial nodes** are the starting execution points of the graphs. There can be only one per graph.
- **Final nodes** end the current execution branch. There can be more than one.
- **Decision nodes** select one execution branch among several, depending on whether the condition for their outgoing edge holds or not. Only the outgoing flow edges from a decision node may have a non-empty condition and a probability less than 1.
- **Fork nodes** split the current execution branch into several parallel branches.
- **Join nodes** integrate several branches back into one, whether they started off from a decision node or a fork node. This is a simplification from UML, which uses different elements to integrate each kind of branch: join nodes and

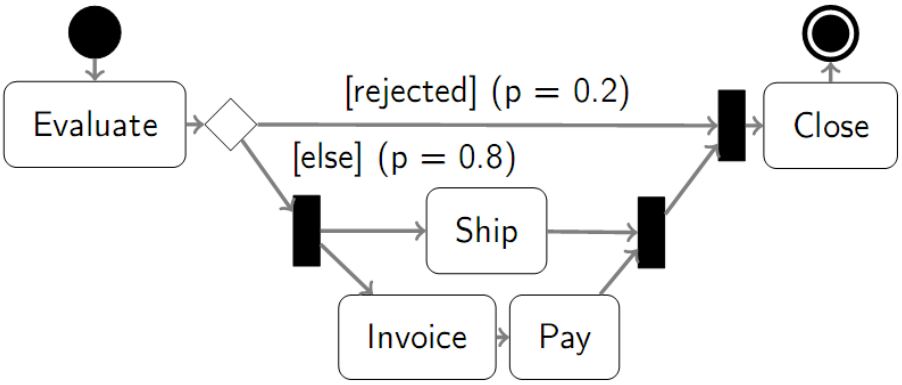

Figure 2. Running example for the inference algorithms

merge nodes, respectively.

3.2. Example model

In order to illustrate the algorithms, we will use the order processing workflow shown in Figure 2:

1. Upon receiving an order, it is evaluated: 20% of all orders are estimated to be rejected, and the remaining 80% are accepted.
2. If the order is accepted, perform these activities in parallel:
   2.1. Create and post a shipping order.
   2.2. Create an invoice and receive payment for it from the client.
3. Close the order.

3.3. Inference of concurrent requests

This first algorithm infers how many requests each task in the workflow must be able to handle per second in order not to become a bottleneck. The algorithm propagates the global performance constraint starting from the initial node, traversing the nodes in ascending order of *depth*. The depth of a node is defined as the length of the longest path from the initial node to it.

For the most part, the number of requests per second (C) is propagated without changes from the initial node. There are two special cases when the propagated value changes:

- Flow edges with probability p lower than 1 propagate pC.
- Join nodes which connect mutually exclusive branches (split off at a decision node) propagate the sum of the values from each branch. Otherwise, the branches split off at a fork node and the minimum value from all branches is propagated. Finding out where the branches split off requires computing the lowest common ancestor of every branch. Using the naïve approach recommended by Bender et al. [12] works best for the sparse graphs commonly used in workflows.

As an example, let us consider the activity "Pay" in Figure 2, when the whole process is required to handle 5 requests per second. This global constraint is propagated mostly unchanged, except for the conditional flow edge from the decision node to the fork node which precedes "Pay". Since its probability is 0.8, "Pay" must handle $0.8 \cdot 5 = 4$ requests per second.

3.4. Inference of time limits

This algorithm infers the time limit for each task from the local and global information available in the workflow, meeting the three following properties:

1. The sum of the time limits in every path from the initial node to any of the final nodes meets the global constraint.
2. Available time is split equally among the nodes, to distribute the load equally across the system. We are also interested in avoiding services that are too coarse-grained or fine-grained.
3. Time limits should be as lenient as possible.

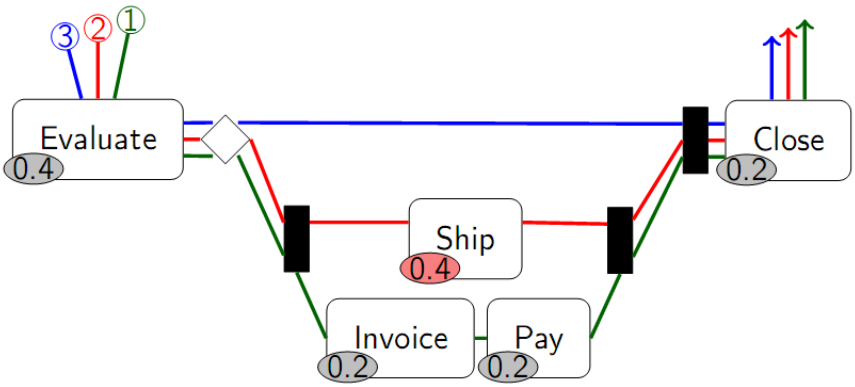

Figure 3. Inference of time limits for the running example

Before describing the algorithm, we need to define this minimal set of terms:

- $F(p)$ is the set of all activities in the path p which lack manual time limits (they are *free*).
- $T_L(G)$ is the time limit for every path in the workflow graph G.
- $T_M(p)$ is the sum of the manual time limits in the path p.
- $S_A(p,G) = T_L(G) - T_M(p)$ is the *slack* left unassigned by the manual times on path p of the workflow graph G.
- $T_E(p,G) = S_A(p,G)/(1+|F(p)|)$ is an estimate of the magnitude of the time limit each task would obtain with only the information of the path p of the graph G. The lower the value, the more restrictive the path is.

The algorithm follows these steps:

1. Previous automatic constraints are removed.
2. Store all paths from the initial node to a final node in L, and sort them in ascending order of $T_E(p,G)$.
3. For each path p in L, check its slack, $S_A(p,G)$:
   - 4.1. If it is negative, that means that the manual times exceed the global constraint. The algorithm is aborted.
   - 4.2. If it is zero, check if there are any free nodes. In that case, we cannot assign a time limit to them, and an error should be reported. Otherwise, continue.
   - 4.3. If it is greater than zero, distribute it equally among the unrestricted activities in the path.

The correctness of the algorithm relies on the ordering defined on the paths: since paths are traversed from most to least restrictive and previously inferred annotations are not overwritten, the algorithm will ensure the global constraint still holds for the previous paths. When all paths are done, the global constraint holds for all of them.

To illustrate the algorithm, let us assume that all requests to the process in Figure 2 should require 1 time unit or less of processing time, and that "Evaluate" is assumed (according to prior knowledge) to take never more than 0.4 seconds. In this case, the bottom path is the most restrictive, followed by the middle and top paths. The bottom path is visited first and the 0.6 seconds of slack are split equally among the 3 unrestricted nodes. The middle path is visited next, and the remaining 0.4 seconds of slack are assigned to "Ship". Finally, the top path has no unrestricted nodes, so we are done.

4. Propagation of performance constraints

In the previous section, we described how the existing global and local information can be used to augment the activities in the workflow with target values for their performance indicators.

In order to describe each activity in further detail, the user can automatically expand them with a new subgraph including a single initial node, a final node and a single subactivity. The local performance constraints in each activity are now the global constraints for their subgraphs.

Therefore, all the user needs to do is repeatedly design the workflows, annotate them, and expand them. Once the high-level business activities have been broken down into simple actions, the process can be executed in a WFMS, using a workflow language such as WS-BPEL 2.0 [13]. These executable actions can be monitored to ensure that the key performance indicators of each high-level

business activity can be reached.

5. Conclusions and future work

Manufacturing firms need to collaborate with others, and to do that they need to formalize their business processes and integrate their information systems. Workflows can model business processes at each level of abstraction. In a service-oriented architecture, the information system is a portfolio of services which can be reused as building blocks for the workflows. However, checking that the desired performance for the business process is obtained from the concrete activities is difficult.

In this work we have presented a top-down approach which ultimately derives the expected performance of the lowest-level activities from the target key performance indicators of the process. Two algorithms fill in the details at a certain level and a transformation lets the modeller proceed.

The algorithms have been successfully implemented as part of the SODM+T toolset [14], which is a set of plug-ins for the Eclipse integrated development environment [15]. The code is freely available under the Eclipse Public License.

However, the algorithms right now only operate on a simplified workflow language, and the time limit algorithm requires all paths in the workflow to be enumerated. We intend to optimize the performance of the algorithms and extend them to popular languages such as BPMN or WS-BPEL 2.0. Later on, the algorithms will allow for finer-grained control of the inference process.

Acknowledgements

This work was co-financed by the Junta de Andalucía. The University of Cádiz is an Academic Associate of the FP6 I*PROMS Network of Excellence on Innovative Production Machines and Systems.

References

[1] K. Johansen, M. Comstock, and M. Winroth, "Coordination in collaborative manufacturing mega-networks: A case study," *Journal of Engineering and Technology Management*, vol. 22, Sep. 2005, pp. 226-244.

[2] C. Piddington, "SME interoperability in the global economy: A discussion paper," *Proceedings of the 16th IFAC World Congress*, P. Zítek, Ed., Czech Republic: 2005.

[3] J. Browne, I. Hunt, and J. Zhang, "The Extended Enterprise (EE)," *Handbook of Life Cycle Engineering: Concepts, models and technologies*, A. Molina Gutiérrez, A. Kusiak, and J.M. Sánchez García, Eds., London, United Kingdom: Kluwer Academic Publishers, 1998, pp. 3-30.

[4] Object Management Group, "Business Process Modeling Notation 2.0 - Beta2," 2010. Available at http://www.omg.org/spec/BPMN/2.0/Beta2/.

[5] OASIS, "Web Service Business Process Execution Language (WS-BPEL) 2.0," Apr. 2007.

[6] R.S. Aguilar-Savén, "Business process modelling: Review and framework," *International Journal of Production Economics*, vol. 90, Jul. 2004, pp. 129-149.

[7] Computer Systems Laboratory of the National Institute of Standards and Technology (NIST), "FIPS 183: Integration Definition for Function Modeling (IDEF0)," Dec. 1993.

[8] T. Erl, *SOA: Principles of Service Design*, Indiana, EEUU: Prentice Hall, 2008.

[9] W3C, "Web Services Glossary," 2004. Available at http://www.w3.org/TR/ws-gloss/.

[10] A. Tharumarajah, A. Wells, and L. Nemes, "A comparison of the bionic, fractal and holonic manufacturing concepts," *International Journal of Computer Integrated Manufacturing*, vol. 9, 1996, pp. 217-226.

[11] Object Management Group, "Unified Modeling Language (UML) 2.2," 2009. Available at http://www.omg.org/spec/UML/2.2/.

[12] M.A. Bender, G. Pemmasani, S. Skiena, and P. Sumazin, "Finding Least Common Ancestors in Directed Acyclic Graphs," *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, 2001, pp. 845-853.

[13] OASIS, "WS-BPEL 2.0 Primer," 2010. Available at http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.html.

[14] A. García-Domínguez, "Homepage of the SODM+T project," 2010. Available at https://neptuno.uca.es/redmine/projects/sodmt.

[15] Eclipse Foundation, "Eclipse.org home," 2010. Available at http://eclipse.org/.