IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, p-ISSN: 2278-8719, www.iosrjen.org
Volume 2, Issue 9 (September 2012), PP 80-83

Authorization for Workflow Processes using Pi Calculus

ANSHULA GARG*, Prof. Pradeep Mishra
[1]*Department of Computer Science & Engineering, Shri Shankaracharya Institute of Professional Management & Technology, Raipur – 492015, C.G., India*
[2]*Shri Shankaracharya College of Engineering & Technology, Bhilai*

Abstract*—Workflow patterns contain basic features of business process. Advanced branching and synchronization patterns present a series of patterns, which characterize more complex branching and merging concepts which arise in business processes. Pi-calculus can be applied in business process modeling.*

This paper describes the applicability of process algebra, the π-calculus, as a formal foundation for authorization in Business Process Management (BPM) for complex organization. We therefore investigate the π-calculus from a technical viewpoint based on current work in this area. The paper summarizes shifting requirements and discusses evolving theories behind BPAC from the past over state–of–the–art to the future. The concepts and theories are concluded by an illustrating example highlighting why the π-calculus is a promising foundation for future BPAC.

***Keywords*—** *BPAC, Petri nets, Pi calculus, BPEL4WS, BPM, CCS, XLANG*

I. INTRODUCTION

The Workflow Patterns Initiative was established with the aim of delineating the fundamental requirements that arise during business process modeling on a recurring basis and describe them in an imperative way (Russell N. A., 2006). There are eight kinds of the workflow patterns, the kind of the most patterns of which is Advanced Branching and Synchronization Patterns. Advanced Branching and Synchronization Patterns presents a series of patterns which characterize more complex branching and merging concepts which arise in business processes.

Mobile systems are made up of components that communicate and change their structure as a result of communication (Puhlmann F. a., 2005). The Pi-calculus is a process algebra that describes mobile systems. Based on the execution semantics of the Pi-calculus, the behavior of each workflow pattern has been defined precisely in (Puhlmann F. a., 2005).

This paper discusses the applicability of process algebra for the description of mobile systems, the π-calculus [1], as a foundation for Business Process Management (BPM). BPM refers to an integrated set of activities for designing, enacting, managing, analyzing, optimizing, and adapting business processes. To distinguish economical from technical aspects, we refer to computerized business processes as workflows. Still, BPAC lacks a uniform official foundation for the technical aspects [2] that might be filled by the π-calculus.

The paper complements the work on the π -calculus as a foundation for BPM from an economical viewpoint as investigated by Smith and Fingar by focusing on technical aspects. We therefore analyze three major shifts in the area of BPM, ranging from system description and distribution aspects up to changing environments in which business processes are executed.

II. PI- CALCULUS

The Pi-calculus is a modern process algebra that describes mobile systems in a broader sense (Milner R. J., 1990). This paper is about implementing the pi calculus language. We therefore start by describing it. To call it a language is in fact a little ostentatious – it is really just a minimal core of commands and declarations, which will apparently be embedded in some richer language. In our current implementation we provide the commands as C functions, and we are working on the richer language; in Biztalk the commands are part of its internal XLANG. We give a brief presentation of the syntax and semantics of the Pi-calculus in this section.

A BPMS should include a *process virtual machine.* This single runtime provides the groundwork for process execution, and such virtual machines are based on the Pi Calculus. This would be interesting but insignificant, were it not for the fact that early implementations of BPMS process engines are able to massively scale, allowing processes to be deployed that rival or exceed those supported today in ERP software. One analyst predicts that such implementations can exceed the performance of existing software systems because existing software relies on separate computational and communicating threads, implemented within heavy-

weight messaging and transaction processing monitors, with the associated overheads arising from the layers upon layers of legacy code and baggage that comes from decades of product evolution and extension, mostly to make up for deficiencies in the legacy itself. By contrast, Pi Calculus engines provide the illusion of communication between participants with no requirement for message passing. Additionally, transactional semantics are inherent in the model, which can ultimately pass up traditional transaction processing altogether. The true benefits of process systems in terms of resource utilization will emerge over the next few years as practical experience grows. They will sooner or later break free legacy systems and demonstrate their value in their own right, as occurred with previous generations of new technologies.

The pi calculus describes programs which run in parallel and which interrelate over channels. Consider the example program

$$\bar{u}x.P \mid u(y).Q.$$

It contains a series $\bar{u}x.P$ ready to send the data x over the channel u, and later on to continue doing P. In corresponding, a second program u(y).Q is ready to receive the data y over the channel, and then continue doing Q. The name y is a formal argument and is local to Q. The two programs interrelate as follows:

$$\bar{u}x.P \mid u(y).Q \rightarrow P \mid Q\{x/y\}.$$

Note how handy it is to write parallel threads and synchronization – easier than with fork, shared data and condition variables, and more flexible than remote procedure calls.

A term may also be replicated, indicated with !u(y).Q. This is a service that runs at channel u and that spawns a fresh copy of Q every time it is invoked. The restriction operator (x)P declares a name x to be local in P. Names are more like heap variables than stack variables: they can be used outside the sub-program in which they were first created, and can persist after the sub-program ends; whereas stack variables can do neither.

A distinguishing feature of the pi calculus is *input mobility*. We explain with an example. When the program $u(x).x(y).Q$ has performed a rendezvous at *u*, then it will wait to perform a second rendezvous at *y* – but the actual channel *y* will only be determined at runtime. 'Input mobility' refers to this late binding of the input channel. It is absent from Biztalk and other distributed calculi such as the join calculus [1], where input channels are known at compile-time. We suspect it will prove to be a convenient feature. For instance, a cell-phone receives the name of a new base station, and then waits for messages from that new station.

III. WHY CONSIDER II -CALCULUS?

The π -calculus and its predecessor CCS (Calculus of Communicating Systems) [5] has strong mathematical foundations for years. This has lead to a glut of deep theoretical results as well as applications in programming languages (e.g. PICT [3]), protocol verification, software model checking (e.g. Zing [1, 4]), hardware design languages2, and several other areas.

Workflow description languages have a lot in common with these areas. First, workflows can be thought of as parallel processes. Second, workflows often defy the block structure found in conventional programming. Third, the outlook of doing formal verification on workflows is very pertinent and using a process calculus makes algebraic reasoning and algebraic transformation instantly possible due to the large body of research already present. Lastly, although they are rather bare in their style, CCS and π -calculus enforce a very strong separation of process and application logic that seems appropriate for authorization in workflow modeling.

The process calculi are highly theoretical constructs that require a great deal of expert knowledge. Workflow description systems should be available to anyone possessing knowledge of the workflow domain being modeled, and so a significant confronts lies in bridging this gap. In particular providing understandable graphical tools and user-friendly abstractions constitutes a pivotal challenge if π -calculus based systems are to succeed or not.

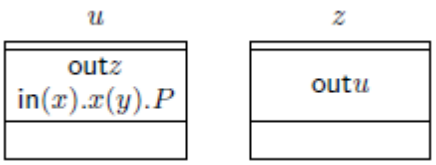
A major strength of the π -calculus is its ability to express passing of channels between nodes and the ability to pass processes. However, only few examples exist that put this capability to use in the workflow domain. It may turn out, though, that channel-passing will become highly relevant when addressing service infrastructure and actual implementation of workflow tasks. By choosing π -calculus we stand a better chance of finding a unified foundation for business systems programming in various fields.

In terms of practical workflow management π -calculus promises seamless integration between workflow systems and existing verification tools. For the end-user this means access to verification tools that will make writing and adapting workflows faster and less error prone.

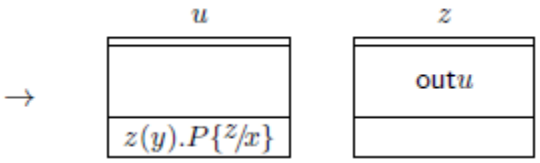
IV. CHANNEL-BASED IMPLEMENTATION

Our distributed implementation of the pi calculus is channel-based. This means that the only things to exist at runtime are channels, each at its own location (or co-located with some other channels). In Biztalk the channels may belong to different companies. We split a program up into packets, each packet ready to

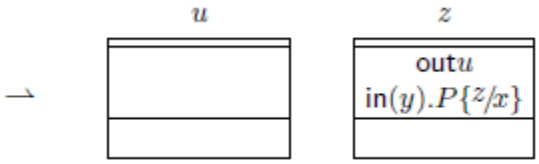
rendezvous at some channel, and we deploy each packet directly to this channel. Thus, synchronous rendezvous is local. We point up our implementation with a small example, and then evaluate with other work.
Consider the program ūz | u(x).x(y).P | ¯zu. After compilation the program is split into three packets, two deployed at *u* and the third at *z*:



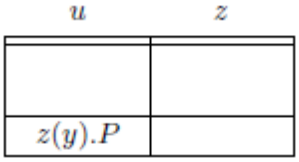
In the diagrams the letters *u* and *z* at the top represent names of channels. Each channel has two areas: the top area contains those packets ready to rendezvous on the channel, and the bottom area (currently empty) contains those packets yet to be deployed. In this example, a rendezvous at *u* is immediately possible:



As a result of the rendezvous, the *continuation z*(*y*).*P {z/x}* has been placed in bottom area of *u*, indicating that it is now ready to be deployed. Deployment involves sendingit across the network to *z*:



As mentioned, we suppose that channels may be co-located in the same address space. We draw such co-located channels as physically adjacent:



If two channels are co-located, then a program packet can be sent from one to the other in constant time. We have used this property to prove efficiency results.

Biztalk/Join solution. Biztalk and the join calculus [1] disallow input mobility, and disallow continuations after the send command *uz*. This means that continuations need never be sent around the system. The join calculus additionally requires that every input command be replicated. This yields the property that every send message is guaranteed to find a ready receiver.

Packets solution. Our solution is to break programs down into smaller fragments, and *pre-deploy* them to their ultimate location. For instance, given *u*(*x*).*v*(*y*).*w* (*z*).*P*, we might break it into *u*(*x*).*v*(*y*) deployed to *u*, and *w* (*z*).*P* deployed to *w*. The result is that the large continuations are already placed at their correct locations; all that is needed is a small message to trigger them. Our machine therefore implements this fusion version of the calculus, rather than the pure pi calculus, and we leave pre-deployment as a compile time optimization.

V. RESULTS AND DISCUSSION

We believe that the pi calculus will prove a good language for writing concurrent and distributed programs: it is simpler than threads, and it looks to be applicable to the full range of programs from low-level device drivers to high-level systems integration. This wide range gives rise to new motivations and new challenges in pi calculus research.

VI. CONCLUSIONS

This paper investigated why traditional formalisms for workflow, like Workflow nets, petri nets are not well suited for future BPAC, as they do not match the shifting requirements of message-based, distributed, and dynamically adapting processes even during emergencies.

Recent results [7, 8] have shown that the π -calculus is well suited for modeling classical workflows, nowadays known as service orchestrations, as well as service choreographies that together form a core foundation of future BPAC based on the shifting requirements. This paper revealed the strengths of mobile systems, i.e. the π -calculus, as a formal foundation for BPAC.

REFERENCES

[1] Martens, A.: Analyzing Web Service based Business Processes. In Cerioli, M., ed.: Proceedings of Intl. Conference on Fundamental Approaches to Software Engineering (FASE'05). Volume 3442 of Lecture Notes in Computer Science., Springer- Verlag (2005).
[2] Christian Stefansen. SMAWL: A SMAll Workflow Language based on CCS. In Proceedings of the CAiSE Forum of the 17th Conference on Advanced Information Systems Engineering, June 2005.
[3] W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. IEEE Intelligent Systems, 18(1):72-76, 2003.
[4] D. Ferraiolo, D. Kuhn, R. Chandramouli, Role-Based Access Control, Artech House, 2003.
[5] D. Sangiorgi and D. Walker. The _-calculus: a Theory of Mobile Processes. Cambridge University Press, 2001.
[6] R. Milner. Communicating and mobile systems: the Pi-calculus. Cambridge University Press, 1999.
[7] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. Rolebased access control models. IEEE Computer 29, 2 (1996), 38–47.
[8] V. Thurner, A formally founded description technique for business processes, Technical Report, Technical University of Munich, Germany, 1997.