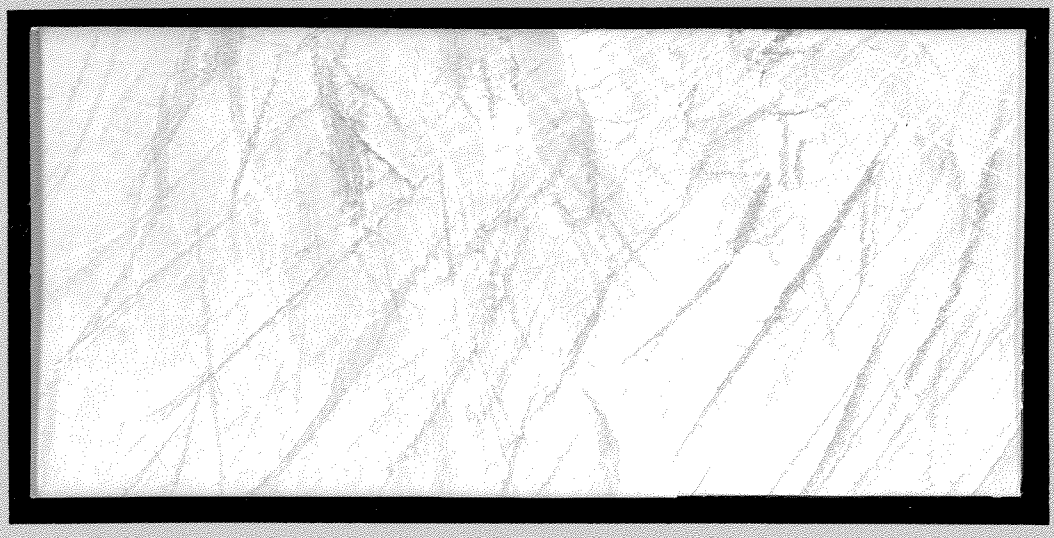
TÜBİTAK

TÜRKİYE BİLİMSEL VE
TEKNİK ARAŞTIRMA KURUMU

THE SCIENTIFIC AND TECHNICAL
RESEARCH COUNCIL OF TURKEY

2001/66



Elektrik, Elektronik ve Enformatik Araştırma Grubu

Electric, Electronics and Informatics Research Grant Committee

TÜRKİYE BİLİMSEL VE
TEKNİK ARAŞTIRMA KURUMU

THE SCIENTIFIC AND TECHNICAL
RESEARCH COUNCIL OF TURKEY

DEĞİŞKEN VE DAĞITIK BİR İŞ AKIŞI YÖNETİM SİSTEMİ VE İMALAT SEKTÖRÜNDE UYGULANMASI

PROJE NO : 197E038

Prof.Dr. Asuman DOĞAÇ

OCAK 2001
ANKARA

Önsöz: Bu projede genel amaçlı değişken ve dağıtık bir İş Akışı Yönetim Sistemi (Workflow Management System) prototipi gerçekleştirilmistir. Proje Avrupa Komisyonu tarafından desteklenen INCO DC 97 2496 "A Workflow System for Maritime Industry (MARIFLOW)" projesi ile paralel yürütülmüştür. Prototip projenin ortakları olan Alman şirketleri tarafından test edilmiş ve başarılı olmuştur.

İçindekiler

Tablo ve Şekil Listeleri

Öz: İş akışı sistemlerinin geliştirilmesini amaçlayan projede denizcilik sektöründeki gemi imalatıyla ilgili şirketlerin arasında döküman akışına bağlı bir iş akışı sistemi oluşturulmuştur. Sistemin prototipi denenmiş ve çok başarılı olmuştur. Sistem tanımlanan iş akışına göre şirketlerin sorumlu olduğu işleri internet üzerinden dağıtmaktadır. Ayrıca güvenlik modülü sayesinde sistemin yetkili olmayan kişiler tarafından kullanılması engellenmiştir. Bu sebeple sistem internet üzerinden güvenli bir şekilde çalışabilmektedir. Sistem tanımlanan iş akışına göre kullanıcıya yapması gereken işi elektronik posta ile göndermekte ve kullanıcı kendi bilgisayarında kurulu olan iş listesi yöneticisi sayesinde sorumlu oldugu işleri görmekte ve bu işler için daha önceden tanımlamış olduğu programları otomotik olarak bilgisayarında çalıştırabilmektedir. Sistemde çıkan herhangi bir aksaklık durumunda sistemin diğer kısımlarıyla ilgili olan iş akısı çalışmaya devam etmektedir.

Anahtar Kelimeler: İş akışı sistemleri.

Abstract: The aim of the project was to develop workflow management systems. In the project, a prototype of workflow management system for the companies related to maritime industry is developed. The prototype of the system was tested successfully. System distributes the responsibilities of companies according to defined workflow over the internet. With the help of the security module, people that are not authorized are prevented to use the system. So, the system runs securely over the internet. According to the workflow defined, the system sends e-mail to the user for the work he is responsible from and the user can see that works from the worklist manager installed to his computer and also he can run any kind of application program he defined before for that work. When any failure at a company, the part of workflow that is related to other part of the system continues to run .

Keywords: Workflow Management Systems.

GİRİŞ

Projede iş akışı sistemlerinin geliştirilmesi amaçlanmış ve çalışmalarımız sonunda dağıtık ve değişken bir İş Akışı Yönetim Sistemi (Workflow Management System) geliştirilerek bu sistem imalat sektörüne uyarlanmıştır. İş akışı sistemleri modern düzenleme yaklaşımları tarafından istenen iş yönetimini destekleyen anahtar bir teknolojidir. İş akışı sistemleri şirketler arasındaki iş kontrolunü sağlamak açısından çok önemli bir yer tuttuğu için son zamanlarda gelişen teknolojiyle beraber akademi ve endüstri bu konuda araştırma ve geliştirmeye yönelmiştir. Ancak günümüzdeki çoğu sistem içerilen işlerle sınırlı ve işlerin nasıl sıralanacağından tamamen bağımsızdır. Başka bir deyişle günümüzde endüstride kullanılan iş akışı sistemleri merkezidir. Bu nedenle bu sistemler bilgi kaynaklarını tanımlama ve internet üzerinden akışını kontrol etme, işleri belirleme ve bu işleri kullanan diğer işleri uyarma gibi mekanizmalardan yoksundur. Bu projede geliştirilmiş olan İş Akışı Sistemi bu eksiklikleri gidermiş ve imalat sektörüne kolaylıkla uyarlanmıştır. Proje konusu uluslarası düzeyde çok büyük öneme sahiptir. Proje süresi içerisinde Mart ayı başında proje ile ilgili bir yayın Amerika'nın San Diego Şehrinde yapılan International Conference on Data Engineering (ICDE) 2000 konferansında sunulmuştur. Ayrıca proje ile ilgili başka bir yayın Eylül ayının ikinci haftasında İsaril'in Eilat şehrinde yapılan Cooperative Information Systems (COOPIS) 2000 konferansında sunulmuştur.

İş akışı sistemlerinin geliştirilmesi amacıyla yapılan çalışmalarımız tamamlanmıştır ve bu çalışmalarımız sonucu elde ettiğimiz bilgiler imalat sektörüne uygulanmıştır.

2. rapor döneminde başlatılan Avrupa Komisyonu destekli "MARIFLOW" isimli proje dahilindeki yoğun çalışmalarımız tamamlanmıştır. MARIFLOW projesinin proje planımızla ilgili en temel özelliği Maritime Konsorsiyumu bünyesindeki denizcilik sektöründeki gemi imalatıyla ilgili şirketlerin arasında döküman akışına bağlı bir iş akışı sisteminin oluşturulmasıdır. Bu özelliği ile hem projenin iş akışı sistemlerinin imalat sektöründe uygulanması kısmına büyük bir uygunluk içermektedir, hem de Avrupa'da endüstriyel ve bilimsel alanda faaliyet gösteren kuruluşlarla birlikte bilimsel ve teknolojik bir çalışma yapılması dolayısıyla projenin başarılı bir şekilde tamamlanmış olmasının Türkiye açısından da büyük faydalar getireceği kesindir. Türkiye'nin Avrupa Topluluğu'na aday olması sebebiyle, MARIFlow ve benzeri projelerin başarılı bir şekilde tamamlanmasının, Avrupa Topluluğu'na üyelik sürecimiz için olumlu bir gelişme olacağı kabul edilmelidir.

Almanya'nın Hamburg şehrinde Mayıs ayının ikinci haftasında ortaklarla yapılan toplantıda sistemin endüstride yaygın olarak kullanılması için gerekebilecek eklentiler üzerine bir değerlendirme yapılmıştır. Bu proje kapsamında geliştirilen sistem üzerine yapılabilecek eklentiler üzerinde çalışmalar devam

etmektedir. Toplantıda bu değerlendirmeler dışında endüstriyel ortakların sitelerine kurulan sistemin test sonuçları görüşülmüştür. Ortaklardan alınan verilerin ışığında sistem üzerinde yapılması gereken değişiklikler not edilmiştir. Ayrıca Hebrew Üniversitesi (İsrail) tarafından geliştirilen güvenlik modülünün sisteme entegre edilmesi tamamlanmıştır.

Mayıs ayının üçüncü haftasından itibaren sistem projenin ortakları olan Alman şirketleri ve tarafımızdan test edilmiş, karşılaşılan aksaklıklar düzeltilmiştir.

Almanya'nın Bremen şehrinde Temmuz ayının ikinci haftasında ortaklarla yapılan toplantıda elde edilen test sonuçları değerlendirilmişir. Bu toplantı sonrasında sistem güvenlik modülü olmadan test modunda çalışabilir bir duruma getirilmiştir.

Ekim ayı içerisinde Avrupa Komisyonu temsilcileri ve projenin diğer ortakları ile yapılan toplantıda projenin genel gidişatı incelenmiş ve temsilciler tarafından proje çok başarılı bulunmuştur.

Aralık ayı içerisinde Almanyanın Bremen şehrinde düzenlenen WONDERMAR isimli iş marketinde projenin gösterimi yapılmış ve çok başarılı bulunmuştur. Aynı gün projenin bitiş toplantısı yapılmış ve proje Avrupa komisyonu temsilcileri tarafından incelenerek çok başarılı bulunmuştur. Ayrıca proje ortakları prototipi ileri düzeyde test etmek için Avrupa komisyonuna projenin 3.5 (üç buçuk) ay daha uzatılmasını taleb etmmişlerdir. Bu taleb şu anda Avrupa komisyonu tarafından incelenmektedir.

Bu proje kapsamında yukarıda belirtilen özelliklere sahip değişken ve dağıtık bir iş akışı yönetim sistemi gerçekleştirilmiş ve kullanıcılar tarafından başarılı bir şekilde test edilmiştir. MARIFLOW projesini genel olarak anlatan yayınlarımız Ek-1 de sunulmuştur:

GELİŞME

Günümüzdeki iş akışı sistemlerinin merkezi olması ve şirketler arsındaki iş ve bilgi akışını kontrol etmekten yoksun olması temel alınarak bu sistemlerin eksiklerini giderecek yeni bir iş akışı sistemi prototipi aşağıdaki veriler ışığında geliştirilmiştir :

- İşlemlerin yerine getirilmesi dağıtık olmalı, başka bir deyişle her bir sitede çalışan iş akışı tanımı üzerinde merkezi bir kontrol olmamalı. Aksi halde kontrolü sağlayan site arızalandığında sistem duracaktır. Bu yüzden sistemi oluşturan parçalar düzenli olarak başka programlar tarafından uyarılmak yerine diğer parçalardan aldıkları mesajlara göre ne yapacaklarına karar vermelidir.

- Sistemi oluşturan parçalar hangi parçalarla haberleşmeleri gerektiğini bilmeli ve haberleşme esnasında durumunu korumalıdır.

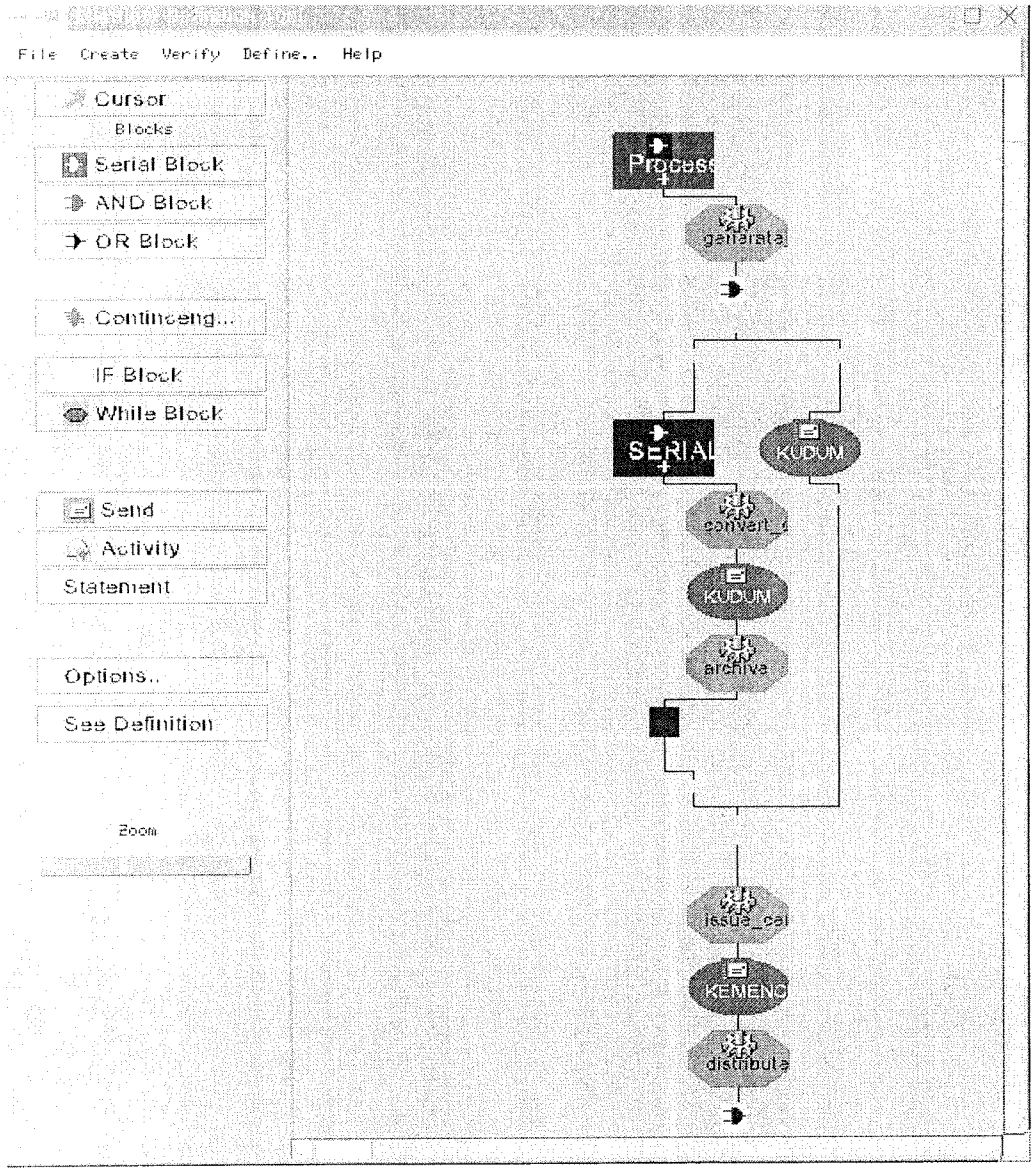
- Sistemi oluşturan parçalar kendi sitelerindeki faaliyetler hakkındaki bilgiyi yerel gözlem ve hatadan sonra tekrar eski haline gelme amacıyla yönetmelidir.

Bu gereksinimler bizi ajan altyapılı bir mimari tasarlamaya itmiştir. Ajanlar karmaşık işleri hiçbir kesintiye gerek kalmadan bağımsız yapabildikleri için diğer yazılım türlerinden ayrılırlar.

Bu bilgiler ışığında tasarlanan sistemde ajanlara ek olarak Grafiksel iş akışı tanımlama programı ve iş listesi yöneticisi kullanılmaktadır, bunlara ek olarak sistemde iş akışını internet üzerinden izlemek için grafiksel bir izleme programı bulunmaktadır. Sistemin parçaları genel olarak şöyledir :

Sistem mimarisi

1. Grafiksel iş akışı tanımlama programı :



Şekil 1 Grafiksel iş akışı tanımlama programı.

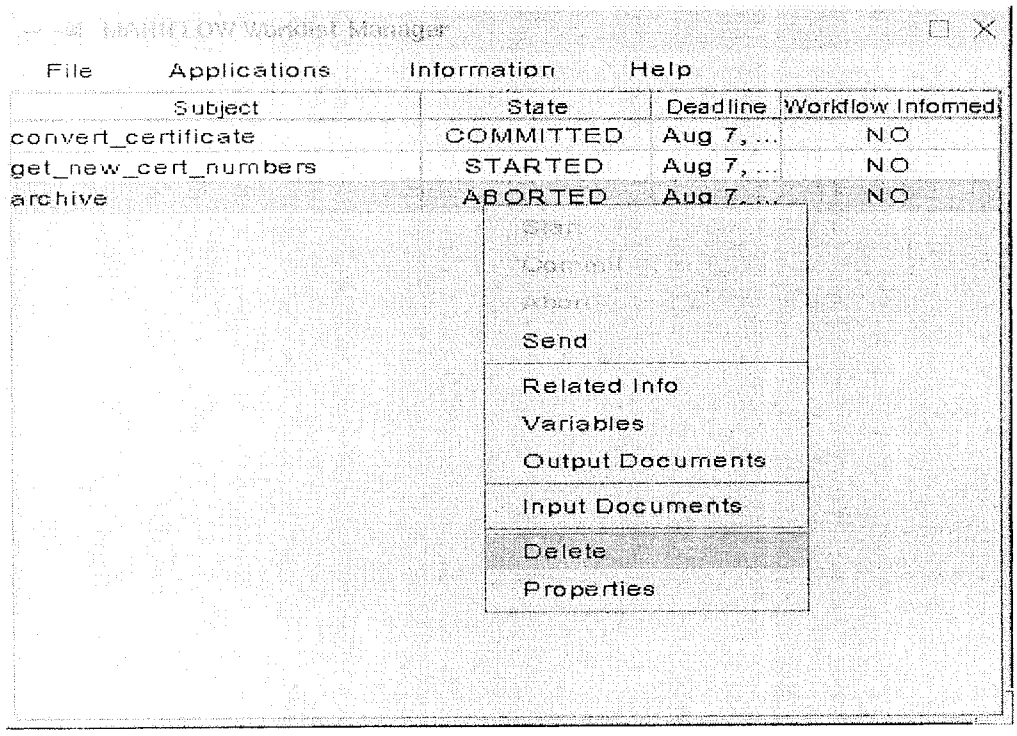
Grafiksel iş akışı tanımlama programı (Şekil-1) sayesinde her türlü iş akışı grafiksel olarak tanımlanabilmekte ve tanımlanan iş akışı, FlowDL diye adlandırılan iş akışı tanımlama diline çevrildikten sonra ajanlar için gereken bilgiler ajanların anlayabileceği bir şekilde internet üzerinden tüm ajanlara güvenlik modülünü kullanarak şifrelendikten sonra dağıtılmaktadır. FlowDL Workflow Management Coalition[2]'da tanımlı olan altı kuralı içermekte ve blok yapısı sayesinde iş akışı tanımını kolaylıkla inşa

edebilmektedir. Ayrıca iş akışının grafiksel şekli izleme programının kullanması için bir bilgi deposuna konmaktadır.

2. Ajanlar

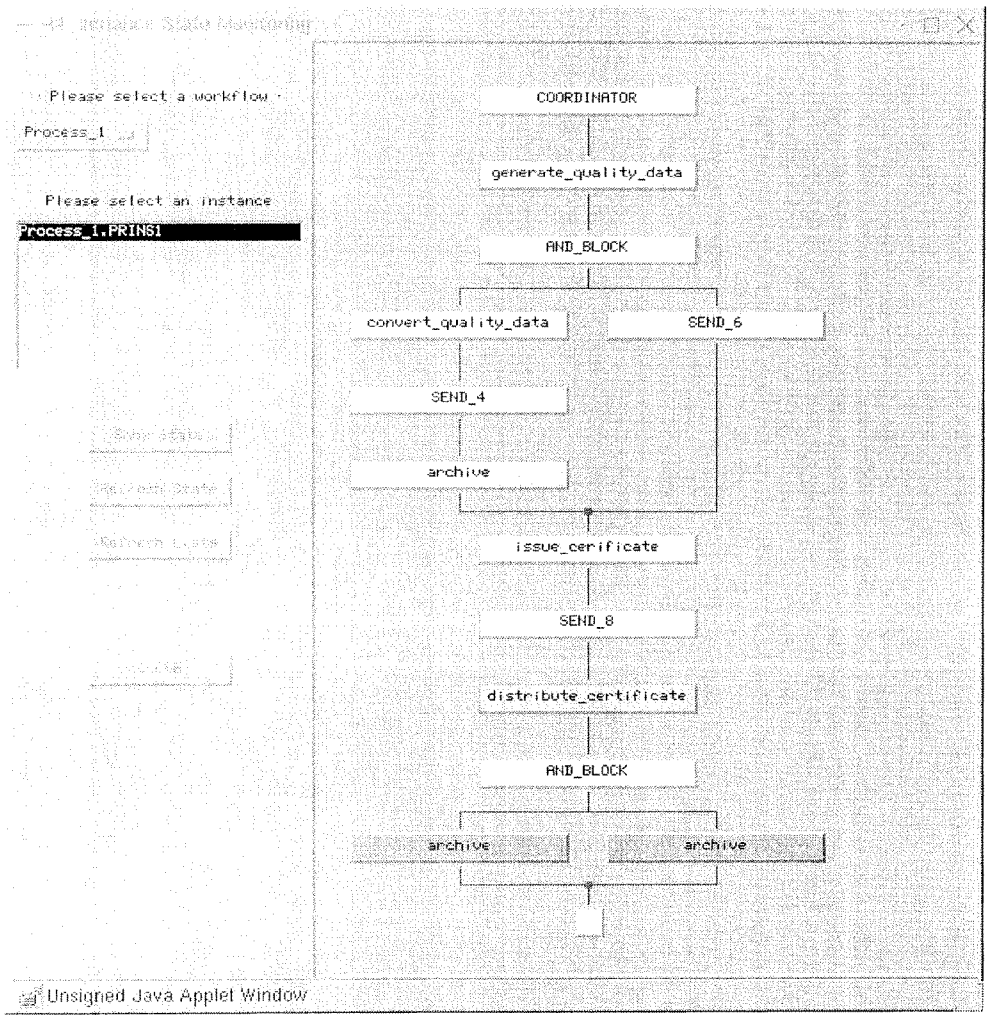
Her ajan ve bu ajana bağlı bulunan kullanıcılar elektronik posta yoluyla haberleşmektedirler. Her ajan kendisine ulaşan iş akışı tanımını ilgili kullanıcıya elektronik posta yoluyla göndermektedir. İş akışı tanımı ajanlar tarafından alındıktan sonra ajanlar bu tanımdan her işlemin başlaması için gereken şartları ve bu şartlar yerine getirildiğinde bilgilendirilecek diğer ajanları liste halinde çıkarırlar. Her ajan diğer ajanlardan aldığı mesajlara göre kendi sitesinde çalışacak olan işlemleri yönetirler. Ayrıca her ajan herhangi bir işlem tamamlandığında, başladığında veya başarısız olduğunda ilgili ajanlara gereken şekilde durumu bildirirler ve iş akışı izleme programı için gereken bilgiyi bilgi deposuna koyar. Elektronik posta yoluyla ajan kullanıcıya hangi işi yapması gerektiğini bildirir. Kullanıcı sonucu yine elektronik posta yoluyla bağlı bulunduğu ajana gönderir. Her kullanıcı kendine ait ajanlarda bulunan iş akışını ajana elektronik posta atarak başlatabilir. Ajan bu postayı aldığında iş akışını başlatmakla yükümlü olan yönetici ajana mesaj gönderir ve yönetici ajan iş akışını başlatır. Ajanlar arasındaki ve ajanlarla kullanıcılar arasındaki tüm mesaj alış verişleri şifrelenmiş bir şekilde yapılmaktadır.

3. İş Listesi Yöneticisi



Şekil 2 İş listesi yöneticisi

İş listesi yöneticisi (Şekil-2) sayesinde kullanıcı tanımlanmış olan herhangibir iş akışını bağlı bulunduğu ajan vasıtasıyla başlatabilmektedir. İş listesi yöneticisi kendisine bağlı bulunan servis programı sayesinde bağlı bulunduğu ajandan gelen elektronik postaları okumakta ve hangi işin yapılacağını öğrenip iş listesine eklemektedir. Kullanıcı herhangi bir iş için kendi makinasında herhangi bir uygulama programını iş listesi yöneticisinden tanımlayabilmekte ve servis programı sayesinde bu uygulama programını kendi kontrolünde veya otomotik olarak başlatabilmektedir.



4. İş akışı izleme programı

Şekil 3 Grafiksel İş akışı izleme programı

İş akışı izleme programı (Şekil-3) internet üzerinden çalışmakta olup sistemde o esnada hangi iş akışlarının tanımlı olduğunu ve tanımlı iş akışlarından kaçar tane başladıgını listeler. Kullanıcı başlamış olan herhangi bir iş akışını secerek o akışın son durumunu grafiksel olarak görebilir.

Diğer iş akışı sistemleri

İş akışı sistemleriyle ilgili bir çok araştırma prototipi ve ticari ürün bulunmaktadır. Bunlardan en önemlilerini şöyle sıralamak mümkündür :

- METUFlow, tamamen dağıtık bir iş akışı sistemi olan METUFlow METU-SRDC de geliştirilmiştir. Geliştirdiğimiz prototipteki iş akışı sisteminin blok yapısı bu sistemden alınmıştır. Ancak METUFlow'un dağıtık çalışma sistemi CORBA tabanlıdır. Bizim gereksinimlerimiz ajan tabanlı sisteme uygundur.

- Exotica/FMDC iş akışı sistemi merkezi olup bağlanmamış kullanıcılar iş akışı yöneticisi olarak kullanılmaktadır. Kullanıcılar sisteme bağlı kalmadan işlerini yapabilmektedir.

- Action Tech Metro, IBM MQ Series, Keyflow ve Ultimus gibi bir çok ticari iş akışı sistemi "prototip gelişen internet teknolojisini kullanarak iş akışı sistemlerinin kapasitesini geliştirmiştir.

SONUÇ

Proje başarılı bir şekilde sonuçlanmıştır. Geliştirilen ajan mimarisi sayesinde varolan iş akışı sistemlerinin kapasiteleri geliştirilmiş ve internet üzerinden değişik şirketler arasındaki komplike iş akışı kontrolünü mümkün kılınmıştır. Proje ile paralel yürütülen Avrupa komisyonu destekli "MARIFlow" isimli projenin başarılı bir şekilde tamamlanmış olması Avrupa Topluluğu'na üyelik sürecimiz için olumlu bir gelişme olmuştur. Ayrıca çıkan yayınlarımızdan birkaçının uluslarası konferanslarda sunulması dolayısıyla projenin ülkemize büyük yararlar getirdiği kesindir. Gerçekleştirilen prototip mevcut iş akışı sistemlerini geliştirip değişik şirketler arasında dağıtık bir iş akışı kontrolünü sağlayabildiğiden ve herhangi bir sektörüne kolayca uyarlanabilirliği açısından ilerisi için geniş bir kullanım alanı olduğu kesindir.

Proje ile ilgili çok ayrıntılı bilgi "Final Report"da sunulmuştur.

Bibliografi

G. ALONSO, R. GUNTHOR, M. KAMATH, D. AGRAWAL, A. El ABBADI, C. MOHAN, (1996), "Exotica/FDMC: A Workflow Management System for Mobile and Disconnected Clients", *Parallel and Distributed Databases*, Vol. 4, No. 3, The Netherlands.

G. ALONSO, C. HAGEN, H. SCHEK, M. TRESCH, (1998), "Towards a Platform for Distributed Application Development", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 195-221.

P. C. ATTIE, M. P. SINGH, A. SHETH, M. RUSINKIEWICZ, (1993), "Specifying and Enforcing Intertask Dependencies", *Proceedings of the 19th VLBD*, Dublin, Ireland.

Q. CHEN, U. DAYAL, (1996), "A Transactional Nested Process Management System", *in Proceedings of 12th International Conference on Data Engineering*, New Orleans.

A. CICHOCKI, M. RUSINKIEWICZ, (1998), "Migrating Workflows", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 339-355.

U. DAYAL, M. HSU, R. LADIN, (1991), "A Transactional Model for Long-Running Activities", *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona.

U. DAYAL, Q. CHEN, TAK W. YAN, (1998), "Workflow Technologies Meet the Internet", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 423-438.

A. DOGAC, E. GOKKOCA, S. ARPINAR, P. KOKSAL, I. CINGIL, B. ARPINAR, N. TATBUL, P. KARAGOZ, U. HALICI, M. ALTINEL, (1998), "Design and Implementation of a Distributed Workflow Management System: METUFlow", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A Sheth (Eds), Springer-Verlag pp. 61-91.

D. GEORGAKOPOULOS, M. HORNICK, A. SHETH, (1995), "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", *Distributed and Parallel Databases*, Ahmed K. Elmagarmid (Ed-in-chief), Volume 3, Number 2, pp. 119-153.

C. HAGEN, G. ALONSO, (1998), "Flexible Exception Handling in the OPERA Process Support System", *18th International Conference on Distributed Computing Systems (ICDCS 98)*, Amsterdam, The Netherlands.

D. HOLLINGSWORTH, (1996), "The Workflow Reference Model", *Technical Report TC00-1003, Workflow Management Coalition*, Accessible via: http://www.aiai.ed.ac.uk/WfMC/

J. MILLER, D. PALANISWAMI, A. SHETH, K. KOCHUT, H. SINGH, (1997), "WebWork: METEOR$_2$'s Web-based Workflow Management System", *Journal of Intelligent Information Systems*, The Netherlands

P. MUTH, D. WODTKE, J. WEISSENFELS, G. WEIKUM, A. DITTRICH, (1998), "Enterprise-Wide Workflow Management Based on State and Activity Charts", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 281-303.

MING-CHIEN SHAN, J. DAVIS, W. DU, Y. HUANG, (1998), "HP Workflow Research: Past, Present, and Future", *in Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 92-106.

A. SHETH, D. GEORGAKOPOULOS, S. JOOSTEN, M. RUSINKIEWICZ, W. SCACCHI, J. WILEDEN, A. WOLF, (1996), "Report from the NSF Workshop on Workflow and Process Automation in Information Systems", *SIGMOD Record*, 25(4):55-67

A. SHETH, K. KOCHUT, (1998), "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 35-60.

M. SINGH, (1996), "Synthesizing Distributed Constrained Events from Transactional Workflow Specifications", *in Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, New Orleans

G. VOSSEN, M. WESKE, (1998), "The WASA Approach to Workflow Management for Scientific Applications", *in Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 145-164.

Ekler

Ek-1. Yayınlar

- Dogac, A., Tambag, Y., Tumer, A., Ezbiderli, M., Hamali, N., "*An Agent-based Workflow System for Inter-enterprise Business Processes*", in Proc. of European Commission's eBusiness and eWork Conference and Exhibition, Madrid, Spain, October 2000.

- Dogac, A., Tambag, Y., Tumer, A., Ezbiderli, M., Tatbul, N., Hamali, N., Icdem, C., Beeri, C., "*A Workflow System through Cooperating Agents for Control and Document Flow over the Internet*" in Proc. Of the Intl. Conf. on Cooperative Information Systems (CoopIS' 00), Israel, September 2000

- Dogac, A., Ezbiderli, M., Tambag, Y., Icdem, C., Tumer, A., Tatbul, N., Hamalı, N., Beeri, C., "*MARIFlow Project*", Intl. Conf. on Data Engineering (ICDE 2000).

- Dogac, A., Beeri, C., Tumer, A., Ezbiderli, M., Tatbul, N. Icdem, C., Erus, G., Cetinkaya, O., Hamali, N., "*MARIFlow: A Workflow Management System for Maritime Industry*", In Proc. of 'Application of Information Technologies to the Maritime Industry', Eds. C.Guedes Soares and J.Brodda, MAREXPO Consortium, 1999, pp. 33-51.

- Koksal, P., Cingil, I., Dogac, A., "*A Component-Based Workflow System with Dynamic Modifications*", In Proc. of Next Generation Information Technologies and Systems (NGITS '99), Springer-Verlag, Lecture Notes in Computer Science, 1649, Israel, July 1999, pp238-255.

- Arpinar, B., Halici, U., Arpinar, S., Dogac, A., "*Formalization of Workflows and Correctness Issues in the Presence of Concurrency*", Journal of Distributed and Parallel Databases, Vol. 7, No. 2, April 1999, pp. 199-248.

- Dogac, A., INCO-DC 97 2496 MARIFlow Project, Final Report.

BİBLİYOGRAFİK BİLGİ FORMU

1. Proje No: 197E038	**2. Rapor Tarihi:** 15 / 01 / 2001
3. **Projenin Başlangıç ve Bitiş Tarihleri:** 15 / 01 / 98 - 15 / 01 / 01	
4. **Projenin Adı:** Değişken ve Dağıtık bir İş Akışı Yönetim Sistemi ve İmalat Sektöründe Uygulanması.	
5. **Proje Yürütücüsü ve Yardımcı Araştırıcılar:** Proje Yürütücüsü: *Prof Dr.* Asuman DOĞAÇ	
6. **Projenin Yürütüldüğü Kuruluş ve Adresi:** ORTADOĞU TEKNİK ÜNİVERSİTESİ, Bilgisayar Mühendisliği Bölümü. İnönü Bulvarı, 06531,ANKARA	
7. **Destekleyen Kuruluş(ların) Adı ve Adresi:** Avrupa Komisyonu, 200 Rue de la Loi, B-1040. Brussels, Belgium	
8. **Özet (Abstract):** İş akışı sistemlerinin geliştirilmesini amaçlayan projede denizcilik sektöründeki gemi imalatıyla ilgili şirketlerin arasında döküman akışına bağlı bir iş akışı sistemi oluşturulmuştur. Sistemin prototipi denenmiş ve çok başarılı olmuştur. Sistem tanımlanan iş akışına göre şirketlerin sorumlu olduğu işleri internet üzerinden dağıtmaktadır. Ayrıca güvenlik modülü sayesinde sistemin yetkili olmayan kişiler tarafından kullanılması engellenmiştir. Bu sebeple sistem internet üzerinden güvenli bir şekilde çalışabilmektedir. Sistem tanımlanan iş akışına göre kullanıcıya yapması gereken işi elektronik posta ile göndermekte ve kullanıcı kendi bilgisayarında kurulu olan iş listesi yöneticisi sayesinde sorumlu oldugu işleri görmekte ve bu işler için daha önceden tanımlamış olduğu programları otomotik olarak bilgisayarında çalıştırabilmektedir. Sistemde çıkan herhangi bir aksaklık durumunda sistemin diğer kısımlarıyla ilgili olan iş akısı çalışmaya devam etmektedir. **Anahtar Kelimeler:** İş akışı sistemi	
9. **Proje ile ilgili Yayın/Tebliğlerle ilgili Bilgiler:** Yayınların listesi ve kopyaları ilişikte sunulmuştur.	
10. Bilim Dalı: Doçentlik B. Dalı Kodu: Uzmanlık Alanı Kodu:	**ISIC Kodu:**
11. Dağıtım [*]	☐ **Sınırlı** ☒ **Sınırsız**
12. Raporun Gizlilik Durumu:	☐ **Gizli** ☒ **Gizli Değil**

Final Report

Project Number:	INCO- DC 97-2496
Title:	A Workflow Management System for the Maritime Industry
Project Acronym:	MARIFlow

Report No: Final	Year: 2001	Month: March
Contract start date: Sept. 15, 1998	Contract termination date: March 31, 2001	
Names of editors and/or authors and their organisations: Prof. Dr. Asuman Dogac Dept. of Computer Eng. Middle East Technical University 06531, Ankara, Turkey		
Report Preparation Date: February 1, 2001		
Report Version: V 1.0		
Classification: Public		

1 Project Overview

In this project we designed and implemented an architecture that provides for automating and monitoring the flow of control and document over the Internet among different organizations, thereby creating a platform necessary to describe higher order processes involving several organizations and companies.

The higher order process is designed through a graphical user interface and mapped to the textual workflow definition language of the system called FlowDL.

A process definition in FlowDL is executed through cooperating agents that are automatically initialized at each site that the process executes. Agents handle the activities at their site, provide for coordination with other agents in the system by routing the documents in electronic form according to the process description. The system is capable of activating external applications (which may be inside the company firewall) when necessary, keeping track of process information, and providing for the security and authentication of documents as well as comprehensive monitoring facilities.

The architecture is general enough to be applied to any business practice where data flow and invocation of activities among different industries and cooperations follow a pattern that can be described through a process definition, however the example application provided in the project is on maritime industry.

1.1 Introduction

Internet has revolutionized the publication of data. More and more enterprises are in need of data exchange over the Internet. This has created a need for a declarative mechanism to define the inter-enterprise data flow where it is possible to define the source of data, its control flow and the activities that make use of this data. By declarative, we mean that the description of the application is in a high level language (or via a graphical user interface).Most of the time it is also necessary to provide for the security of the documents as well as keeping track of them for monitoring purposes.

In this work we describe a system that provides for automating and monitoring the flow of control and data over the Internet among different organizations, thereby creating a platform necessary to describe higher order processes involving several organizations and companies.

The architecture is general enough to be applied to any business practice where data flow and invocation of activities among different industries and cooperations follow a pattern that can be described through a process definition. However the example application provided is on maritime industry since the system is implemented within the scope of the MARIFlow project, namely "A Workflow Management System for Maritime Industry".

In the case of maritime information systems the involved organizations include the steel companies, ship yards, classification societies and legal/insurance/ government establishments and a classical example is the certification process. Materials used in ship building or repairs need to be certified by a classification society. In current practice, the material is checked while it is at the production plant and the "quality data" is delivered to the classification society. If the quality data fulfills the requirements, a paper certificate is issued and delivered to the production plant as well as to the customer. Once issued, the certificate follows the material to the main shipyard or to one of the subcontractors from where it will eventually be added to the ship's documentation file. The certificate is checked at every production stage as well as at the ship's handover and at each survey during ship's life cycle.As in any industry involving the flow of large amount of paper documents among different organizations, this is a slow, expensive, tedious, error-prone and very limiting process which in some cases, can hinder the ability to improve the service quality.

In the MARIFlow system, the higher order process is defined through a graphical user interface which is then mapped to a textual language called FlowDL. FlowDL is a block structured language encapsulating the six primitives defined by the Workflow Management Coalition through its blocks with which it is possible to describe flows and hence construct a workflow specification. FlowDL allows to indicate the source of the documents, their control flow and the activities that make use of these documents.

A MARIFlow process is executed through cooperating agents, called MARCAs (MARIFlow Cooperating Agents) that are automatically initialized at each site that the process executes. MARCAs handle the activities at their site, provide for coordination with other MARCAs in the system by routing the documents in electronic form according to the process description, keeping track of process information, and providing for the security and authentication of documents
as well as comprehensive monitoring facilities.

This agent based architecture of the workflow system is necessitated by the following application requirements:

•The execution of the higher order process should be fully distributed. In other words each site executing parts of a workflow definition should act without a central control since a centralized control will create a serious bottleneck, i.e.,when the site, which exerts control over different parts of the workflow, becomes inaccessible due to a crash or a link failure, the whole system will come to a complete stop. Hence the "components" executing at each site should accept messages and decide on their own on how to proceed rather than being synchronously invoked by other programs. In this way, when a site fails only that site and the sites waiting messages from that site are effected.

•The "components" should find out about the other "components" that they need to communicate with, and preserve their state during communication. As an example a "component" should know when and whom to send a document.

• "Components" need to manage the information about the activities at their site both for local monitoring purposes and for recovering from failures.

These requirements necessitate an agent based architecture.Agents are distinguished from other types of software because they are independent entities capable of completing complex assignments without intervention, rather than as tools that must be manipulated by a user.

Specifically agents in MARIFlow have the following properties:

1.They accept messages rather than being invoked by other programs.

2.They evaluate the messages that they receive and act in response rather than being told what to do. i.e., they are autonomous.

3.They find out about the other agents that they need to communicate.

4.They keep their state while communicating with other agents.

The responsibilities of the agents (MARCAs) in our architecture are as follows:

•A MARCA receives messages through a persistent queue and evaluates them to decide what specific action to take.

•It persistently stores the documents it receives. It should be noted that the organizations may be reluctant to grant access inside the corporate firewall. In such cases when the need arises, the MARCA passes these documents to an in-house system by properly acknowledging the in-house system on further processing that may be necessary on the documents. The MARCA is also responsible for getting the documents from the in-house system and forwarding them to the related agents as specified in the process definition.

•Process related information also needs to be stored persistently for monitoring purposes. In our system MARCAs store the information related with monitoring to any JDBC compliant database to be accessed through a JDBC interface.

•There is a single MARCA at each site that handles all the activities of all workflow definitions related with that site. Therefore a new MARCA is generated only for a site participating to a workflow definition for the first time.

If we summarize, the functionality provided by the system developed is as follows:

•A declarative means to specify the control of document flow over the Internet where it is possible to define the source of data, its control flow and the activities that make use of this data.

•Authentication and security of documents and the process related information.

•A monitoring mechanism for keeping track of the documents and/or for providing detailed account of the current status of a process instance within the system.

•Measures for failure recovery and exception handling

This section of the report is organized as follows: In Section "Architecture", the general architecture of the system is described. This section introduces the FlowDLworkflow definition language, the MARIFlow Cooperating Agents (MARCAs), initialization of the system, communication between agents and monitoring of the workflow processes. Section "Authentication" describes how security and authentication of documents and messages are handled in MARIFlow. The failure and exception handling issues are discussed in Section "Availability". Finally, Section "Conc" concludes .

1.2 The Architecture of the System

1.2.1 An Overview of the Architecture

Figure 1 shows an overview of the general architecture of the MARIFlow system. Each organization may have in-house applications inside a firewall protected from unauthorized accesses. MARCAs exist outside the firewall and inform in house applications when necessary through a User MARCA Interfacing Application (UMIA). A coordinating MARCA is installed in one of the sites.

In MARIFlow, an inter enterprise workflow is defined graphically as shown in Figure 2. This tool allows the workflow designer to specify domains, tasks and process information which are then used in building the process definition graphically. This definition is mapped to the textual FlowDL language. The information on sites at which a MARCA should be installed are obtained from the domain definition in the process specification. These sites download a generic MARCA from a given URL. At compilation time the guards of activities within the responsibility of a MARCA are also determined according to the process definition and the MARCAs are initialized with these guards through the coordinating MARCA. Guards are logical expressions for significant events of activities of a MARCA like "start" and "terminate". MARCAs evaluate these guards with the messages that they receive to decide on their actions. As an example, the start guard of an activity handled by a MARCA can be the arrival of a document, say, "doc1" from site "S1" and a document "doc2" from site "S2". In this case, MARCA will start execution of this activity when this AND expression evaluates to true by the arrival of the mentioned documents. Clearly the guards are generated from the information given in the process specification. Similarly "terminate" guard of an activity handled by MARCA may require the transmission of a document, say, obtained from the in-house system to another MARCA. It should be noted that this transmission is realized through persistent queues to survive through crashes.

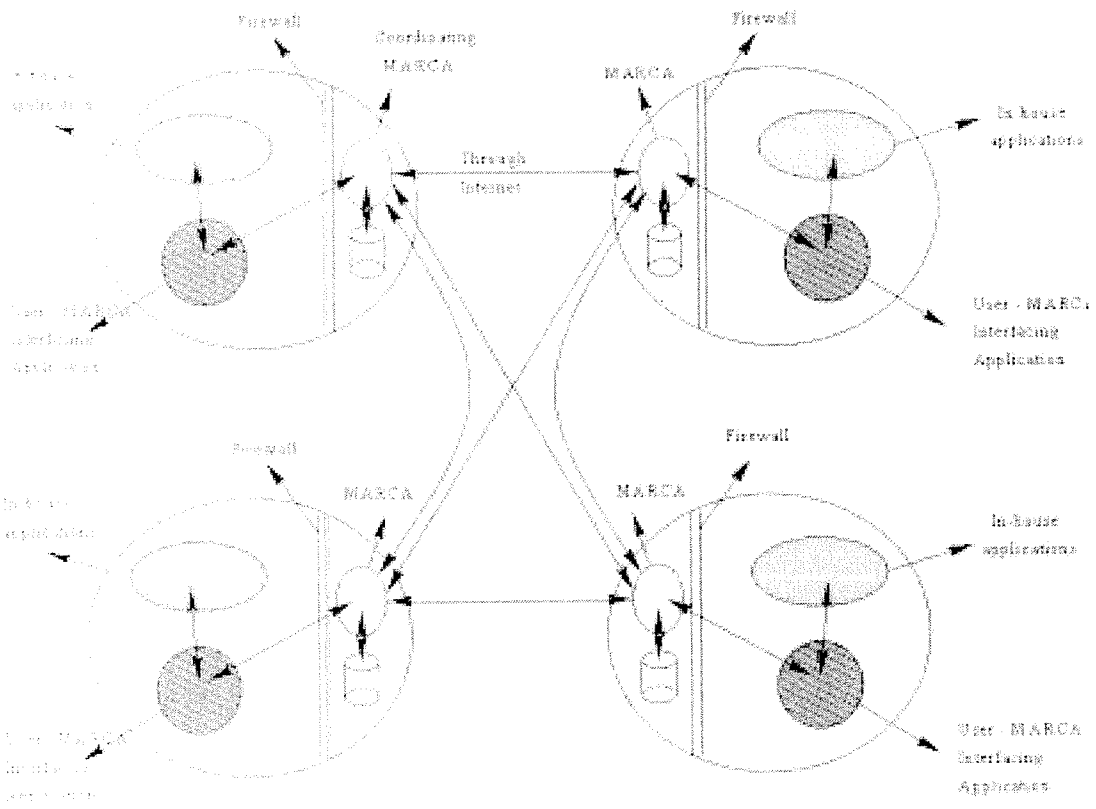


Figure 1:An Overview of the Architecture

1.2.2 Workflow Definition Language: FlowDL

A workflow process is defined in FlowDL as a collection of blocks, tasks and other subprocesses as well as some explicit declarations and commands to specify Internet domain addresses, sources of documents, process specific information to be used for monitoring document flow and activities that need to do further processing on the documents or external activities that need to be invoked. The term activity is used to refer to a block, a task, or a (sub)process.

FlowDL contains seven types of blocks, namely, serial, and_parallel, or_parallel, xor_parallel, contingency, conditional and iterative blocks. These block types encapsulate the workflow primitives defined by WfMC which are sequential, AND-split, AND-join, OR-split, OR-join and repeatable task. The serial block implements the sequential primitive. And_parallel block models the AND-split and AND-join primitives. OR-split, OR-join pair is modeled by or_parallel block. Conditional block corresponds to OR-split and OR-join primitives. Finally, repeatable task primitive is supported by the iterative block.

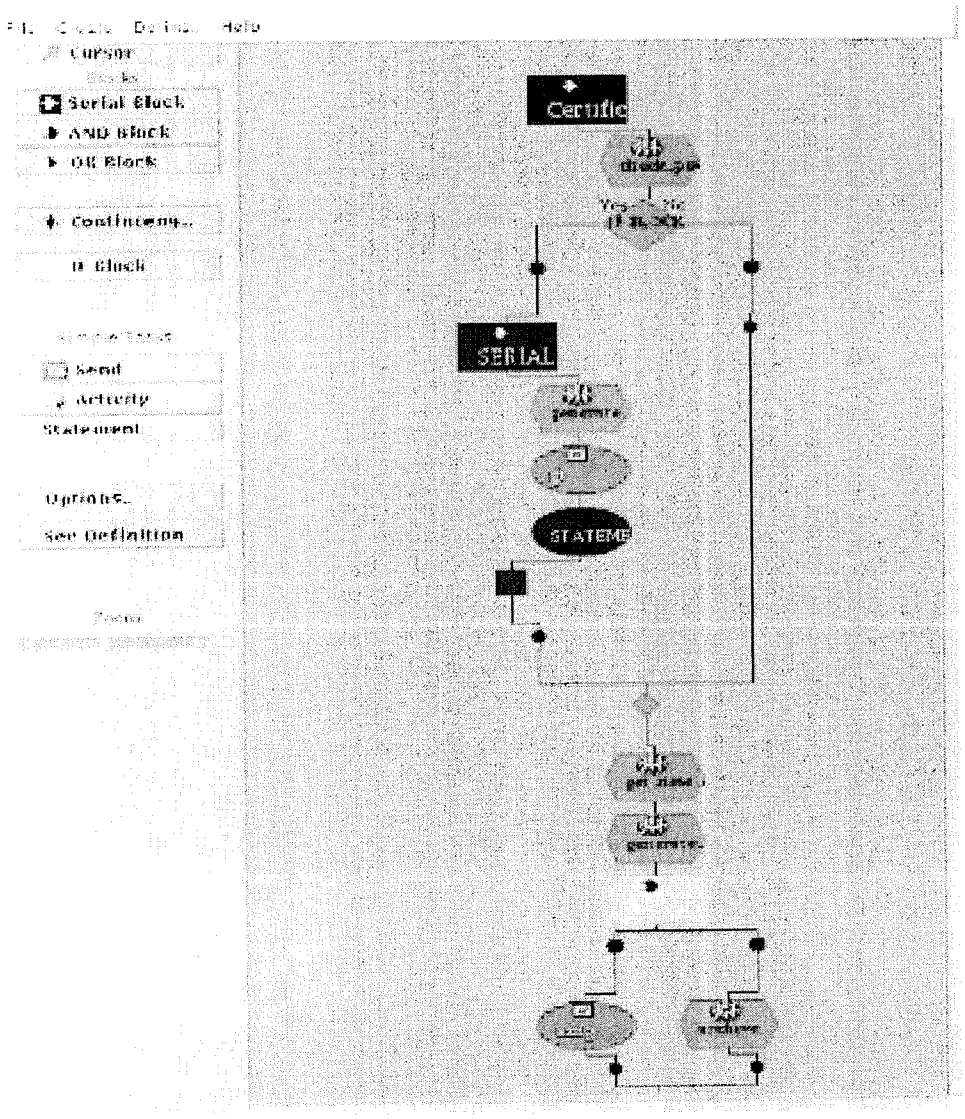


Figure 2: The Graphical User Interface for Process Definition in MARIFlow

The advantages brought by this language can be summarized as follows:

•As noted in nsf, state-of-the-art workflow specification languages are unstructured and/or rule based. Unstructured specification languages make debugging/testing of complex workflow difficult and rule based languages become inefficient when they are used for specification of large and complex workflow processes. This is due to the large number of rules and overhead associated with rule invocation and management. FlowDL avoids these disadvantages through its block-structured nature.

•A block structured language confines the intertask dependencies to a well formed structure which in turn proves extremely helpful in generating the guards of activities for distributed scheduling of a workflow.

•A block clearly defines not only the data and control dependencies among tasks but also presents a well-defined recovery semantics, i.e., when a block fails (aborts), the tasks that are to be compensated and the order in which they are to be compensated are already provided by the block semantics.

In the following we formally define the block semantics:

Syntax 1 $B = (A_1;A_2;A_3;.....;A_n)$, where B is a serial block.

Definition 1 Start of a serial block B causes A_1 to start. Commitment of A_1 causes start of A_2 and commitment of A_2 causes start of A_3, and so on. Commitment of A_n causes commitment of B. If one of the activities aborts, the block aborts.

Syntax 2 $B = (A_1 \& A_2 \& ..... \& A_n)$, where B is an and_parallel block.

Definition 2 Start of an and_parallel block B causes start of all of the activities in the block in parallel. B commits only if all of the activities commit. If one of the activities aborts, the block aborts.

Syntax 3 $B = (A_1 \mid A_2 \mid ..... \mid A_n)$, where B is an or_parallel block.

Definition 3 Start of an or_parallel block B causes start of all of the activities in the block in parallel. At least one of the activities should commit for B to commit but B can not commit until all of the activities terminate. B aborts if all the activities abort.

Syntax 4 $B = (A_1 \parallel A_2 \parallel ..... \parallel A_n)$, where B is an xor_parallel block.

Definition 4 Start of an xor_parallel block B causes start of all tasks in the block in parallel. B commits if one of the activities commits, and commitment of one activity causes other activities to abort. If all of the activities abort, the block aborts.

Syntax 5 $B = (A_1,A_2,.....A_n)$, where B is a contingency block.

Definition 5 Start of a contingency block B causes start of A_1. Abort of A_1 causes start of A_2 and abort of A_2 causes start of A_3,and so on. Commitment of any activity causes commitment of B. If the last activity A_n aborts, the block aborts.

Syntax 6 $B = (condition,A_1,A_2)$, where B is a conditional block.

Definition 6 Conditional block B has two activities and a condition. If the condition is true when B starts, then the first activity starts. Otherwise, the other activity starts. The commitment of the block is dependent on the commitment of the chosen activity. If the chosen activity aborts, then B aborts.

Syntax 7 $B = (condition;A_1;A_2;.....;A_n)$, where B is an iterative block.

Definition 7 The iterative block B is similar to serial block, but start of iterative block depends on the given condition as in a while loop and execution continues until either the condition becomes false or any of the activities aborts. If B starts and the condition is true, then A_1 starts and continues like serial block. If A_n commits, then the condition is reevaluated. If it is false, then B commits. If is true, then A_1 starts executing again. If one of the activities aborts at any one of the iterations, B aborts.

Syntax 8 A = (A_c, AbortList(A_c)), where A_c is the compensation activity of A.

Definition 8 The compensation activity A_c of A starts if A has committed and any of the activities in AbortList(A_c) has aborted. AbortList is a list computed during compilation which contains the activities whose aborts necessitate the compensation of A. If both an activity and its subactivities have compensation, only the compensation of the activity is used. If only the subactivities have compensation, it is necessary to use compensations of the subactivities to compensate the whole activity.

Syntax 9 T = T_u, where T_u is the undo task of task T.

Definition 9 The undo task T_u of T starts if T fails.

In addition to activities, there are also assignment statements in FlowDL which access and update workflow relevant data. In this way the workflow designer has the ability to assign values to variables.

We have also observed that certain activities, like the archival of a document by a MARCA may take long to execute or may fail. This may cause unnecessary delays or even unsuccessful termination of the whole process. To handle such cases we provide the workflow designer with the ability to define activities with a NON_VITAL attribute. Non vital activities are assumed to terminate successfully as soon as they start. However if they fail, they raise an exception. For example for a document archival, if this activity is declared as non vital, the workflow can continue assuming that the document is archived as soon as it is invoked. If the activity fails, an exception is raised and the exception handler makes sure that the document is properly archived. As a summary, the functionality provided by the non vital activities is to delegate certain tasks to the exception handler so that it can run in parallel with the workflow process itself. It should be noted that a workflow process instance will terminate only when all the exceptions raised during its flow are properly handled.

The following is an example workflow defined in FlowDL reflecting a business scenario for a certification process in maritime industry:

```
PROCESS MARIFlow ();

ACTIVITY archive (IN document arch_document);
```

```
ACTIVITY convert_qlty_data(IN document quality_data
        OUT document convted_q_d);
ACTIVITY convert_certificate(IN document certificate
        OUT document convted_cert);
ACTIVITY issue_certificate (IN document conv_quality_data
        IN document product_spec OUT document certificate);
ACTIVITY delete_from_archive();
ACTIVITY cancel_converted_data();
ACTIVITY cancel_certificate();

DOMAIN_DEFINITION      {
        salzgitter_ag.de   szag;
        germanlloyd.org    gl;
        balance_bremen.de  bal;
        isisanisi.com      isisan;
}

struct process_info    {
        string  order_no;
        string  material_no;
        string  customer_name;
        string  supplier_name;
        string  class_society_name;
        string  certificate_number;
}

DEFINE_PROCESS MARIFlow ()
{
szag SENDS (quality_data) TO bal GENERATES (order_no
        material_no customer_name supplier_name
        class_society_name);
AND_PARALLEL {
SERIAL {
        START convert_qlty_data (IN quality_data OUT
        convted_q_d) AT bal COMPENSATED BY
cancel_converted_data();
bal SENDS (convted_q_d) TO gl AND isisan;
START archive (IN conted_q_d) AT isisan NON_VITAL
COMPENSATED BY delete_from_archive();
}
isisan SENDS  (prod_spec) TO gl;
}
START issue_certificate (IN convted_q_d IN prod_spec OUT
certificate) AT gl GENERATES (certificate_number)
COMPENSATED BY cancel_certificate();
gl SENDS (certificate) TO bal;
```

```
START convert_certificate (IN certificate OUT convted_cert)
AT bal;
bal SENDS (convted_cert) TO szag AND isisan;
AND_PARALLEL {
START archive (IN convted_cert) AT szag NON_VITAL
COMPENSATED BY delete_from_archive();
START archive (IN convted_cert) AT isisan NON_VITAL
COMPENSATED BY delete_from_archive();
  }
}
```

Example 1. An example Process Definition for the Maritime Industry

The process starts when the steel company (szag) sends the "quality data" to another company (bal) to be transformed into EDIFACT (Electronic Data Interchange for Administration, Commmerce and Transport) standard Then within the scope of an AND_PARALLEL block, the following two activities are executed in parallel:

•At bal, the in-house system within the firewall is alerted so that the convert_qlty_data process is activated to do the conversion into EDIFACT standard. The document produced as a result (that is, the converted quality document) is sent to gl and isisan. At isisan, the in-house system is alerted to archive the incoming document. These activities are executed within the scope of a SERIAL block since termination of one (e.g., "bal SENDS convted_q_d TO isisan") signals the start of another (e.g., "START archive AT isisan"). However the archival of converted quality data at isisan should not delay the activity that follows (e.g., "START issue certificate AT gl") therefore it is declared as NON_VITAL. Otherwise the block concept requires the archival activity to execute to completion. The completion of the SERIAL block together with the activity "isisan SENDS prod_spec TO gl" will complete And_Parallel block so that the "START issue_certificate AT gl" can start.

•The steel user (isisan) sends the product specification document to the classification society (gl).

Once gl receives the converted quality data, its in-house system is notified to start the "issue_certificate" process. When the issued certificate is delivered to the MARCA outside the firewall, it is transferred to bal again for conversion to the EDIFACT standard and this document is sent to szag and isisan. Finally activities to archive these documents at szag and isisan are executed in parallel.These activities may also be declared as NON_VITAL in which case their
successful termination becomes the responsibility of the exception handler which handles the failure by possibly attempting it several times before invoking a manual user task as explained in Section 4.

A MARCA sends a document and process related information inside the firewall through an e-mail as an e-mail attachment. Since the content of documents may be binary

and thus may include 8-bit data, and since e-mail protocols send only 7-bit data, a base64 encoding is applied to the e-mail attachments.

There is a User MARCA Interfacing Application (UMIA) and a server inside the firewall, namely the Application Server. The Application Server reads the incoming mails and informs the UMIA accordingly. In order to invoke external applications automatically within the MARIFlow System, a mapping should be provided from the activities defined in the workflow definition to the external applications in the company domain. This mapping is achieved through a graphical user interface which is a part of the UMIA that also allows parameters of the application to be specified. If the invocation is to be done manually, i.e., no mapping is defined, then the user invokes the task from the UMIA manually as shown in Figure 3

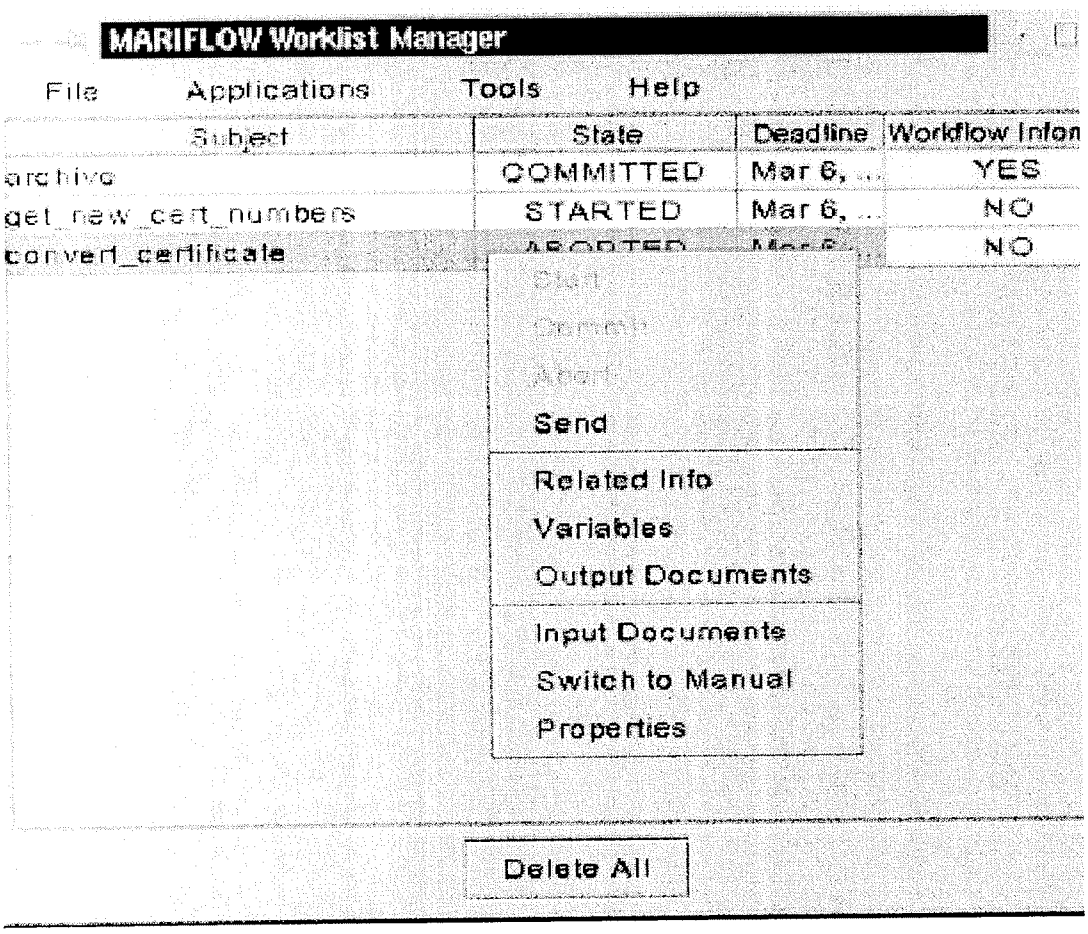


Figure 3: GUI of User MARCA Interfacing Application(UMIA)

1.2.3 MARIFlow Cooperating Agents: MARCAs

A workflow instance in MARIFlow is executed by cooperating agents called MARCAs. There is exactly one MARCA at each site participating to the workflow

execution and it handles all the activities running at its site. Activities start, fail or terminate according to some guards and conditions and these guards and conditions are evaluated through messages exchanged among MARCAs. In other words, a MARCA receives the messages sent to it by other MARCAs, evaluates the guards and the conditions of the significant events (start, fail or terminate) of activities under its responsibility and for each of these significant events determines which other MARCAs should be informed. All of this information is obtained from the process definition during its compilation and is sent to a MARCA during the initialization of the system once for a given workflow definition.

A workflow definition is realized through the Workflow Definition Tool and compiled only once. During this compilation the information mentioned above is obtained from the definition and used in the initialization of MARCAs. Since a single MARCA exists at each site and handles all the instances of any workflow definition there is a need to uniquely identify both different workflow definitions and process instances. The process instances of the same workflow definiton are differentiated in the system through unique process instance identifiers automatically assigned by the system. Different process definitions are differentiated through unique type identifiers, again automatically assigned by the system.

The information handled by a MARCA per activity instance is depicted in Figure 4. In this figure, the "occurred events queue for start" depicts, for each activity, the messages received (and hence the events happened) for this activity to start. These messages on happened events as they are received continuously simplify the guard tree until it becomes true.

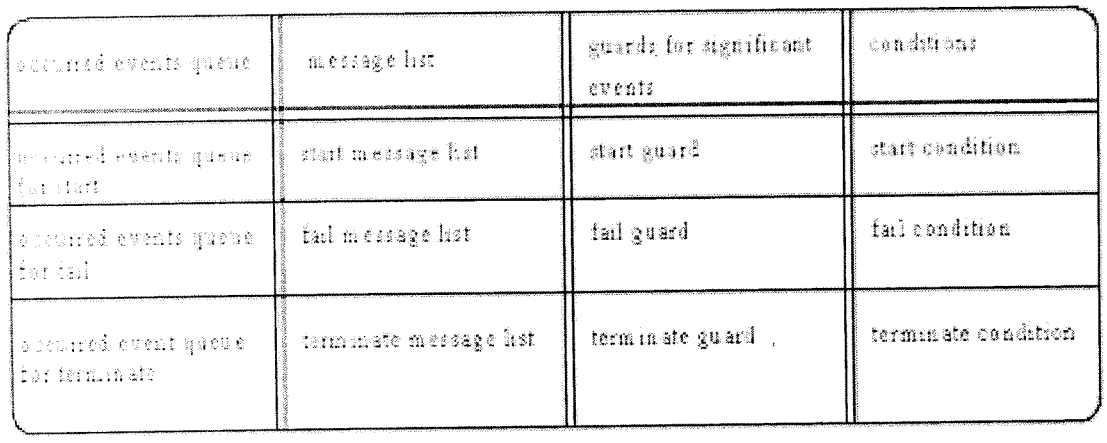


Figure 4: Information Handled by a MARCA

The other queues similarly handle the significant events "fail" and "terminate". "start message list" gives the list of activities (and hence the MARCAs) to be informed when the current activity starts, and "fail message list" and "terminate message list" similarly indicate the activities to be informed when the current activity fails or

terminates, respectively. "start guard" together with "start condition" determines when this activity will start. Note that guards are simplified through occurred events.

A detailed formal description of obtaining guard expressions from a given workflow specification is given in [5]. In this section how MARCAs cooperate is explained through Example 1.

When FlowDL for Example 1 is compiled a process tree is generated as shown in Figure 5. The guards corresponding to the process definition is given in Table 1. When a process is activated an instance conforming to this definition is created. The start guard of activity 1 (SENDS activity at szag) which sends the quality data to bal becomes true. The MARCA at szag starts this activity if start condition is also true which is the availability of the quality data at this MARCA's database. When this activity successfully terminates it informs activity 2 and the start guard of the activity 2 (AND_Parallel block) becomes true. Activity 1 knows that it will inform activity 2 of its termination through its "terminate message list". Note that there is no start condition related with this activity. The start of activity 2 is informed to the activity 3 which is handled by the coordinating MARCA and activity 7 which is handled at isisan.

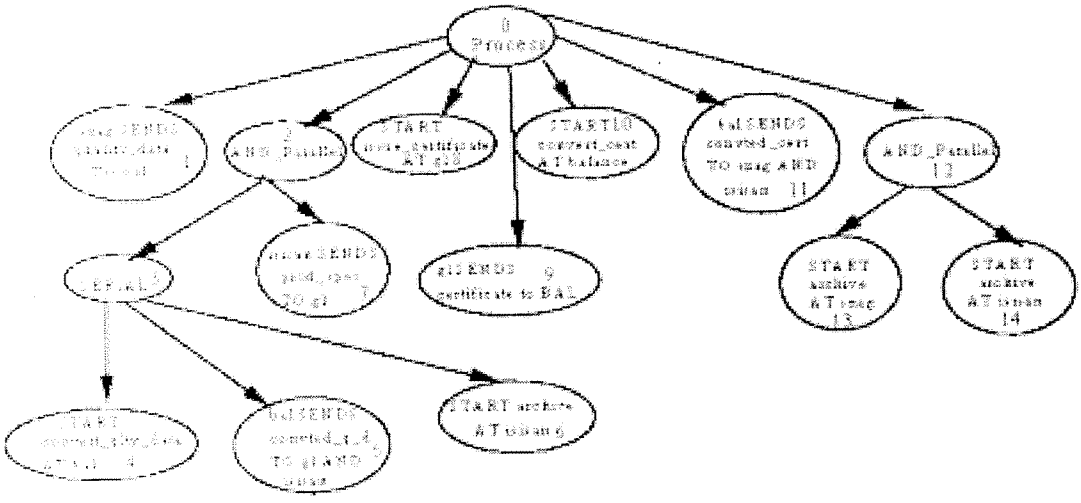


Figure 5: Process tree of the Example Process Definition

Table 1: Guards of the Agents for Example Workflow Definition

Label	Start Guard	start condition	fail guard	fail condition	terminate (successful) guard	terminate condition
0	TRUE				12 terminates	
1	0 starts	quality_data ready	TRUE		TRUE	
2	1 terminates		3 fails OR 7 fails		3 terminates AND 7 terminates	
3	2 STARTS		4 fails OR 5 fails OR 6 fails		5 terminates	
4	3 starts	quality_data ready	TRUE		TRUE	
5	4 terminates	converted_q_d ready	TRUE		TRUE	
6	5 terminates	converted_q_d ready	TRUE		TRUE	
7	2 STARTS	prod_spec ready	TRUE		TRUE	
8	2 terminates	converted_q_d ready AND proc_spec ready	TRUE		TRUE	
9	8 terminates	certificate ready	TRUE		TRUE	
10	9 terminates	certificate ready	TRUE		TRUE	
11	10 terminates	convted_cert	TRUE		TRUE	
12	11 terminates		13 fails OR 14 fails		13 terminates AND 14 terminates	
13	12 starts	convted_cert	TRUE		TRUE	
14	12 starts	convted_cert	TRUE		TRUE	

Activity 3 through its message list knows that it should signal its start to activity 4. As it is clear from this explanation, MARCAs know which messages to send and to whom and the messages received by a MARCA is used in evaluating the guard expressions to decide when a significant event of an activity will happen.The information content for a MARCA processing activity 3 is shown in Figure 6. In this figure, normally the terminate guard of activity 3 is "6 terminates". But since this activity is declared as NON_VITAL, its termination guard is "5 terminates". The execution of the process continues in this way. Note that the guards of events shown as TRUE in Table 1 indicates that these events may occur any time.

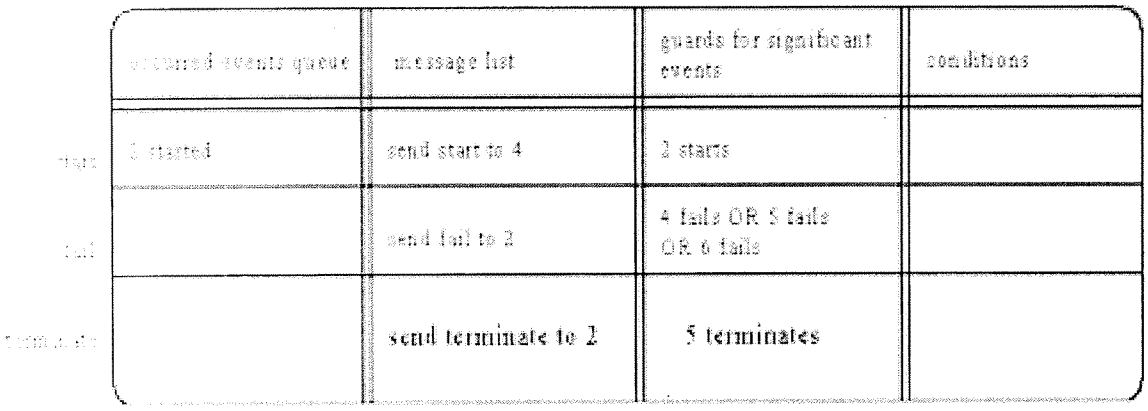


Figure 6: Information Content of a MARCA processing Activity 3

Each activity in the process tree receives/sends messages from/to other activities through its parent block. Carrying block semantics to the execution reduces the number of messages to be communicated. As an example, assume that we have a process segment like:

```
serial {
        and_parallel {
                T1();
                T2();
                ...
                Tn();
                    }
        and_parallel {
                T1();
                T2();
                ...
                Tn();
                    }
}
```

Without a block abstraction during execution, the start guard of each activity in the second and_parallel block must contain the commit event of each task of the first and_parallel block. Obviously this necessitates to communicate the commit event of each of the n tasks in the first and_parallel block to each of the n tasks in the second and_parallel block. Hence without a block abstraction, the number of messages to be communicated is n^2, as shown in Figure 7(b).

When block abstraction is used during execution as shown in Figure 7(a), the start guard of the second and_parallel block contains the commit event of the first and_parallel block. Thus the commit guard of the first and_parallel block contains the commit events

of each of its n tasks; the start guards of each of the tasks in the second and_parallel block contain the start event of the second and_parallel block. For this case, the number of messages to be communicated reduces to 2n + 1, as shown in Figure 7(a). It should be noted that there is a coordinating MARCA in the system which is responsible for executing the block activities.

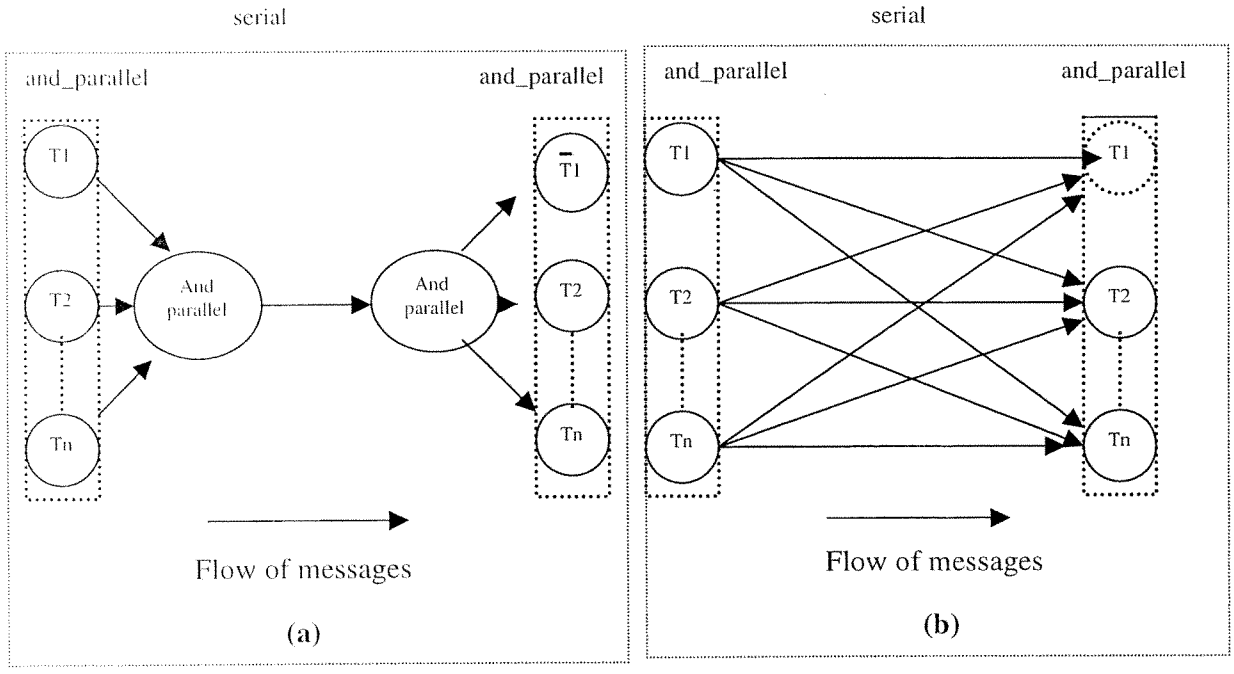


Figure 7: Environment of objects (a) with and (b) without block abstraction

1.2.4 Initialization of the System

Before cooperating agents can be used within a workflow process, they need to be downloaded and setup at a site which will participate in that process. This download operation is necessary only for new sites that will join the execution. MARCA templates are generic java classes, which, when downloaded from a Web site, are executed and they wait for initialization on a pre-defined TCP/IP port.

As the first step the MARCA reads the configuration file. This file contains the port number that the MARCA will communicate through, the type of the JDBC connection to use to connect to the local database, the e-mail addresses of itself and its corresponding UMIA are defined.

Second step for MARCA initialization is the transmission of the related process information like the guards and the message lists from coordinating MARCA to all other MARCA templates which will participate in the workflow process. This information is

transmitted in a communication language specific to MARIFlow, which contains the following:

•Host name where MARCA resides. This information is also necessary for attaching a host name to some of the messages where it is necessary to distinguish the same message coming from different hosts, for example Send1.germanlloyd.org.terminate and Send1.isisanisi.com.tr.terminate.

•Activity label is passed with every message to identify the activity in a MARCA that will accept the request.

•Start, Fail and Terminate message lists. These lists contain the addresses of other MARCA's which will be informed of the significant events (start, terminate, fail) of the currently executing MARCA. Each list consists of a remote MARCA's host, a related ActivityID, and a flag representing Start, Fail or Terminate guard of a remote MARCA.

•Start or Terminate conditions. There may be a logical expression that needs to be true for a guard of a significant event of an activity to fire. Conditions hold these logical expressions and may also contain the list of document names whose existences are required for the condition to evaluate to TRUE. Just before an activity starts or terminates, its MARCA checks the related condition. If the condition, for example requires the availability of a certain document and the document is available, the activity resumes. Otherwise it runs a daemon process that informs when the related documents become available. In case of a general failure of document availability, an exception is raised which is handled by an exception handler. The exception handler may check a predefined number of times and after a timeout period, may invoke a manual user activity.

•Start, Fail and Terminate guards of activities to be executed by the MARCA at this site. The guards are expressions and therefore they are stored in guard trees which persistently reside in a database.

1.2.5 AGENT Communication

MARIFlow agents (MARCAs) communicate with each other through TCP/IP over the Internet. We preferred to use TCP/IP over HTTP and FTP for its efficiency. Network sockets are the basic mechanism of communication among MARCAs. For each MARCA one predefined socket is assigned to be used during the initialization and the communication between the other MARCAs.

It should be noted thata MARCA is downloaded only once to a site and it handles all instances of any workflow definition. In other words, once created the MARCA stays alive until it is explicitly killed. If a crash occurs, the MARCA is recovered as explained in Section 4.

In a communication session, each message is preceded by an activity identifier, that differentiates the activities. An instance identifier is also added to each message since there may be several workflow instances in different states of execution for the same task. This information is also necessary to identify which document belongs to which process instance. If there are several workflow definitions they are assigned specific workflow id's automatically by the system which are used to distinguish task names for different workflow definitions within a MARCA.

Concurrent accesses to a single port is handled by Java's Net Package by assigning dynamic ports for each request. Simple message buffering and queuing are also provided by this package. These queuing facilities have been extended to persistent queue implementation and transactional agent communication for safe and consistent transmission.

MARCAs communicate with each other through an Agent Communication Language, which is specific to MARIFlow agents. That is, agents in MARIFlow system communicate with each other but not with agents in outside world. This is because communicating with outside world agents is not necessary within the scope of the work.

1.2.6 Monitoring of Workflow Processes

Each MARCA stores all the documents and messages it receives persistently in its database. Since Java is used in coding the MARIFlow system, a Java native JDBC interface is used in accessing the database system. Consequently any database with a JDBC interface can be used with a MARCA. The information stored also serves the purposes of a log, that is, after a crash, a MARCA can be brought back to a consistent state by using this information.

Each MARCA also stores additional process related information specified by the workflow designer through FlowDL. For the Example 1, this information includes order_no, material_no, customer_name, certificate_number, etc. Note that which activity is responsible for providing a specific piece of information, such as order_no, or certificate_number is obtained through the GENERATES statement of the activities. In the case of Example 1, the activity "START issue_certificates (IN convted_q_d IN prod_spec OUT certificate) AT gl GENERATES(certificate_number);"declares that certificate_number will be produced by this activity.

Each MARCA sends a copy of the information it stores to the central database for monitoring information. This site is available to any authorized user on the Internet through a Web interface and may further be replicated for availability purposes.

The monitoring facilities provided by the MARIFlow system are twofold:

•When an authorized user provides the process identifier (supplied to her by the MARIFlow system when the process starts), s/he can track the flow of that process instance through a graphical interface as shown in Figure 8.

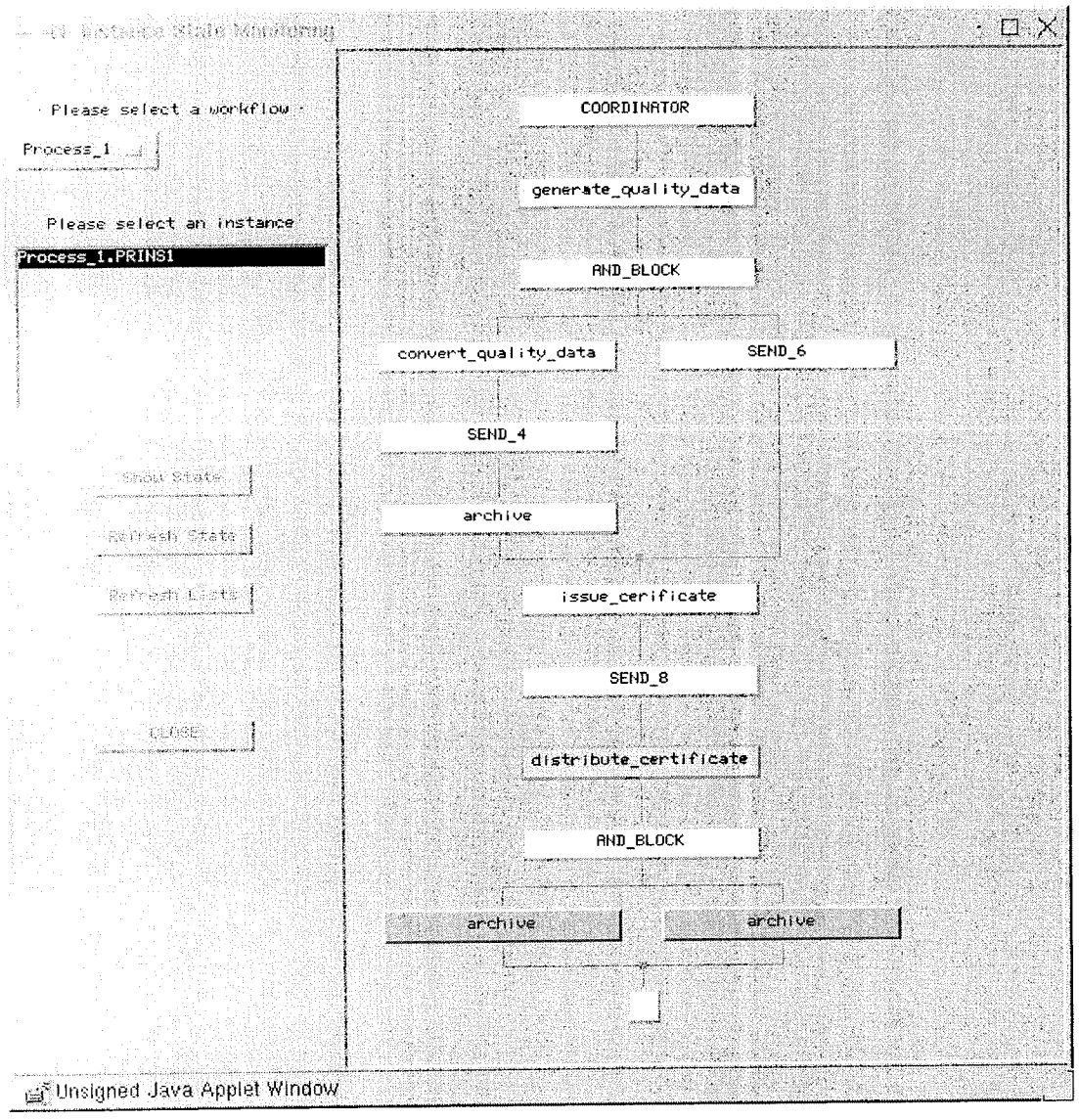


Figure 8 : Graphical Monitoring Interface of the system

•All the information in the coordinating MARCA's database can be queried through SQL query language. In the case of Example 1, some example queries that can be readily answered by the database include: Status of a certificate given its number, status of all certificates for a given steel order, number of certificates issued by the classification society. Figure 9 depicts a a customized GUI currently available for the MARIFlow project.

1.3 MARIFlow Security Services

Within the life cycle of a process definition an application in the company domain prepares a message, then passes it to local UMIA. It, in turn, passes it to the corresponding MARCA. The MARCA sends the message to one or more other MARCAs according to the process definition. The receiver MARCAs pass the messages to their corresponding UMIAs, which pass them to the local applications. Thus, inter/intra company communications basically consists of communication sessions between MARCAs and between a MARCA and its UMIA.

The security requirements of the system can therefore be summarized as follows:

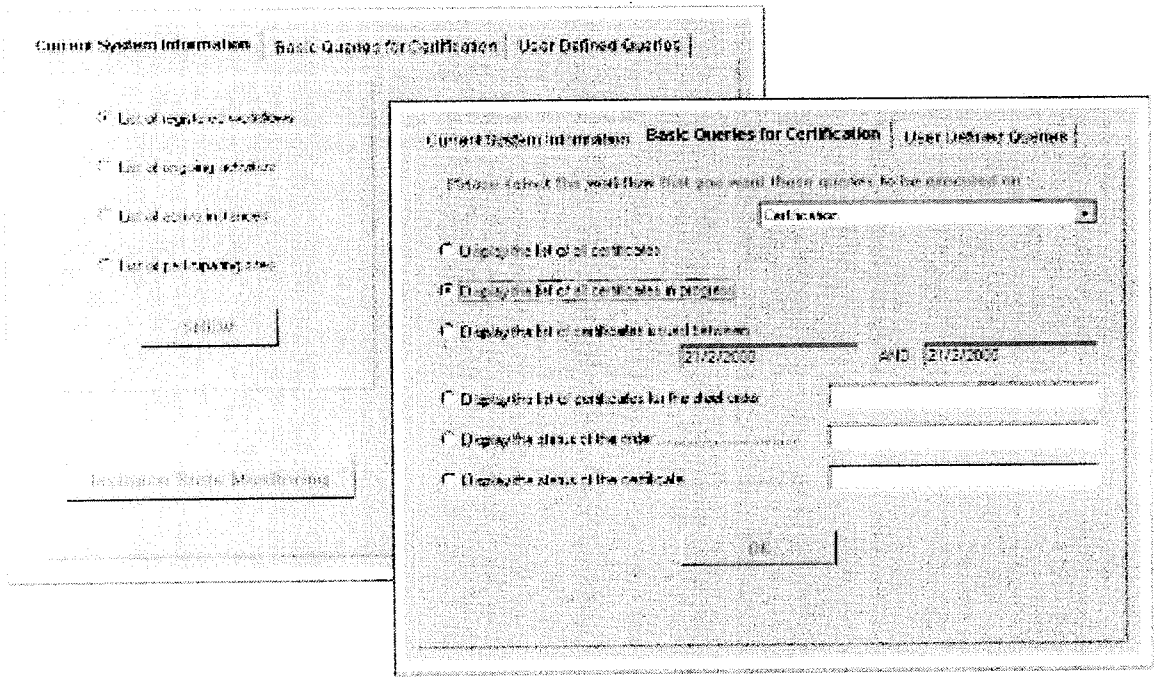


Figure 9: Monitoring Interfaces customized for the Certification Process

•Confidentiality of data in transfer between MARCAs: The contents of a message, including the process description fields, should not be visible to anybody except the sender and the receiver.

•Confidentiality of data in transfer between a MARCA and the corresponding UMIA, as above.

•Confidentiality of data transfer for inter/intra-company communications. Here, even the MARCA software, and probably also the UMIA, is excluded from reading the message. The process description fields, however, are visible, to enable the UMIA and MARCA to decide on how to handle the message.

•Authentication and integrity for all three levels.

•Signatures for some of the inter/intra-company messages, such as certain types of test and certification document. Here, a message consists of a document that is passed in order to be stored.

Note again that a message sent inside a company, as part of a MARIFlow process, should not be readable by MARCA software, or by anybody, except by the sender and receiver inside the company. Thus, confidentiality, authentication and integrity exist in the system at two levels: user to user, and UMIA-MARCA-MARCA-UMIA.

1.3.1 Security Mechanisms

In the following paragraphs, the common mechanisms that areused in security systems to achieve such requirements are described, in order to provide the necessary background to understand the MARIFlow Security Architecture.

Message digest (aka digital fingerprint) is a one-way function (essentially a hash function) applied to a message. It is a short summary of the message, so that its transfer or storage is cheap. The probability that two messages produce the same digest is very small. Hence, when a message and a digest that is claimed to be produced from that message, are given, if applying the same function to the message gives the same digest, then with a very high probability it is indeed the same message. In other words, the message has not been tampered with, for otherwise it would not produce the same digest.Symmetric cipher: This refers to a class of approaches to encryption and decryption that use the same key for both directions. That is, a key is used for encrypting the message. The same key is also used to decrypt it.Asymmetric cipher: This also refers to a class of approaches. The basic idea is to use a pair of keys for each participant. Each key defines a transformation on messages, and the transformations corresponding to the two keys are inverses of each other. One key is made known to all participants, and no effort is made to keep it protected in any way. It is referred to as the public key. The other key, called the private key, is known only to its owner. In general, the sender encrypt the message with the public key of the known receiver. Upon receival of the message, the receiver uses its own private key to decrypt the message. In another mechanism the sender encrypts the message with its own private key and sends the message. The receiver decrypts the message with the public key of the sender. Hence in both techniques authentication is achieved.

For a digest to fulfil its mission, namely guarantee the integrity of a message, the digest itself has to be reliable, i.e. the receiver has to be sure that it was produced from the original message. The possibility that the message has been replaced, a new digest prepared from the new message and then the new message and the new digest presented to the receiver has to be ruled out. In other words a digest can provide integrity, but this has to be accompanied with authentication and integrity of the digest itself. Regarding the

asymmetric keys, the availability of public keys that need not be protected, and are known to all, makes many tasks easier. However encryption/decryption using such a system is expensive, much more so than using a symmetric key. The cost also depends on the key length, but short keys are less secure. Symmetric ciphers are more efficient in use. Therefore the accepted wisdom is that private-public key systems should be used only for short strings. Confidentiality for large messages is achieved by using symmetric keys for the encryption/decryption. The use of a symmetric key for confidentiality has an associated danger: If the enemy listens long enough, and collects many message encrypted with the same key, then breaking the key may become easier. The security can be achieved if the symmetric key used to encrypt messages is often replaced, since then even if an adversary makes an effort to discover the key, he may not have a sufficient number of messages to enable him to make substantial inferences about the key. The key used in this technique is called the Session Key. Even if the session key is eventually discovered, that is of little utility, since the key is soon to be or has already been replaced by another. A typical scenario is that a symmetric key is used only for one communication exchange, or only for one session (hence the name), or a short time period (e.g., day, week etc.)

The security architecture of the MARIFlow System is given in the following.

1.3.1.1 Steady State of the System

The steady state of the system is the state when the components have all the information they need to communicate with each other. This information is the following: Each UMIA and each MARCA has a pair of public/private keys. Each component knows its own private key. Each MARCA knows the public keys of all other MARCAs, and of its corresponding UMIA. A UMIA knows both the public and the private keys of itself and of the corresponding MARCA. Each MARCA and UMIA has a name, referred to as an alias, such that a MARCA knows the aliases of all other MARCAs and of its UMIA. The keys are known to each participant in the form of key-alias pairs. In the MARIFlow System, it is the case that a UMIA does not send messages to specific UMIAs - a UMIA does not have the information about the process definition. Hence, there is a need for an alias that represents all UMIAs, and also a pair of keys for this virtual UMIA. Encryption of a message by a UMIA is done using this virtual UMIA as a receiver, and each UMIA can decrypt the message, since all UMIAs know the keys of this virtual UMIA.

During encryption, the encryptor first generates a 128 bit symmetric session key, which it uses to encrypt the input text. The session key is then encrypted, using the receivers public key. This guarantees that only the receiver can decrypt and find the session key thus provides confidentiality. To provide integrity and authentication, a message digest is prepared, and the digest is encrypted with the senderÆs private key. This implies that any receiver can authenticate the sender of the digest, by using the public key of the claimed sender to decrypt the digest. Consequently, nobody else can forge the digest. The digest itself, with its authentication and integrity guaranteed, can now be used to assure the authentication and integrity of the message. The sender alias,

the encrypted session key, the encrypted signature, and the encrypted text are concatenated into one 64 base string which is actually the whole message that will be sent.

After the arrival of a message, the receiver recovers the session key using the receiverÆs private key. Then the message is decrypted, using that session key. The digest is then decrypted, using the senders public key, thus ensuring its source and integrity. The digest function is applied to the message, and the ensuing digest is compared to the given one, thus verifying the authenticity and integrity of the message. The return value is the open message

1.3.1.2 Initialization of the System

The participants, namely the MARCAs and the UMIAs, and their aliases are known and fixed throughout a process definition. A trusted party exists in the system, whose alias and public key are known to all participants. That trusted party generates a pair of public-private keys for each participant and encrypts each pair using his private key. Public keys are sent to all participants that need to know them, similarly encrypted to prevent forgery. At each partner site the MARCA is informed of the necessary keys for the given workflow. The key-pair of a UMIA and public keys of the virtual UMIA and the associated MARCA are sent to the UMIA via e-mail and the steady-state scenario has been reached.

1.4 Failure and Exception Handling

Workflow processes are long-running activities consisting of many nested sub-activities. Many of the activities within the scope a workflow instance may commit (i.e., make their changes permanent) during its execution and therefore when a workflow instance fails, already committed activities need to be undone to erase the effects of an unsuccessful instance. Compensation activities are used for this purpose to logically undo the effects of the activities in case of failures.

Another concern for failure recovery in the MARIFlow system is the recovery of MARCAs from failures and these issues are discussed in this section.

1.4.1 Recovery of MARCAs

When a site goes down, restarting the MARCA is under the responsibility of the Operating System's start up control. The site's start up control is to analyze the persistent logs of the MARCA and start a new instance using these stable logs created before the site crash. However, there is need for a further mechanism to prevent any Operating System related problem.

In Mariflow, for each MARCA installed there is a background process at that site, called the "rescue process". The rescue process is responsible for monitoring the life time of the agent and checks the MARCA at specific time intervals through a predetermined socket. A thread of the MARCA listens to this socket and responds to the signals. If the MARCA does not respond to this process for a given period of time, the process starts sending signals more frequently. If the MARCA still does not respond, after sending a bunch of signals the process assumes that the MARCA is not functional. The two possibilities in this case are: the MARCA could be blocked or it could be dead. When the rescue process is unable to find the OS process that belongs to this MARCA (i.e., it is dead), it instantiates a new MARCA by the help of the persistent logs related with the state of the agent.

Otherwise if the MARCA is blocked, it is necessary to kill the old instance prior to installation of a new instance. Since the logs are persistent it is possible to recover the state of the MARCA killed and hence the site does not suffer from any inconsistencies.

For the described mechanism to work correctly it is necessary to make sure that rescue process stays alive. Therefore, just as the rescue process checks to see that the MARCA stays alive, the MARCA also checks to ensure that the rescue process stays alive by signalling the rescue process at predefined time intervals. It is MARCA who reinstantiates the rescue process when it dies.

1.4.2 Compensation

Effects of terminated activities of failed workflow instances in MARIFlow are undone through compensation activities. If there is a compensation activity for the whole block this one is used. Otherwise the compensation activities of the involved activities are used to roll back the block. When a parallel activity is to be aborted; the activities currently in execution in each branch receives a signal and the branches are compensated.

Compensation activities in the MARIFlow system are defined using the Process Definition Tool during workflow definition and it has the following form:

```
START Activity(IN some_document OUT another_document)
        AT a_MARCA_site
COMPENSATED BY Compensation_activity
```

Since workflow processes are long-running activities aborting a workflow process instance for a failed activity is usually not desired. It is necessary to roll back the instance to a stable state and restart the execution possibly through an alternate track (roll forward).

A hierarchical approach to failure handling described in [3] allows for partially rolling back the workflow instance to the nearest point in process history tree where it is

possible to restart the execution. When a sub-activity T fails, it is necessary to determine the impact of that failure on the ancestors of T by finding out the highest level ancestor that should be aborted. The root of the activity sub-tree to be logically undone upon T's failure is called the Logical-Undo Root (LUR) of T. Every activity in an activity hierarchy has a corresponding LUR, which may be one of the following [3]:

•the closest non-vital ancestor since its failure can be ignored by its parent,

•the closest ancestor with a contingency activity (children of a contingency block),

•the closest ancestor without a parent, which may be the top-level process or a compensation activity.

Bottom-up searching for LUR in the process tree constitutes the first phase of the approach. After the LUR is found, the effects of the activities in the subtree with LUR as the root are removed in a top-down fashion. Applying the undo operation top-down provides a timely reaction to a failure by halting the activities in scope of LUR promptly. In this second phase, starting from the LUR, the finished activities are compensated (if they have compensation activities) taking the semantics of the blocks that enclose them into consideration. The approach also allows compensations to be made as high level as possible since compensating a high-level activity is more general than compensating a lower-level activity. After this two-phased algorithm is applied to the activity hierarchy, the execution restarts and rolls forward from the next activity after LUR. It should be noted that the full log of every instance is kept by the system and used for recovery purposes.

1.4.3 Exception handling

There is a need to handle the exceptions that may arise during the execution of the workflow process instances. An exception is an unusual event, erroneous or not, that is detectable either by hardware or software and that may require special processing. The type of exceptions handled by the MARIFlow System are as follows:

1.4.3.1 Semantic Exceptions

Semantic exceptions occur when a deviation from the expected behaviour in the program logic is encountered like when a flight booking activity can not find any available seats in a flight. The alternative actions can be specified by testing the returned value by this activity within the scope of an IF block. A contingency activity may serve the same purpose by giving an alternative course of activities when an activity fails; however IF block also allows some conditions to be tested while specifying alternate actions. An example to the use of contingency activity in the Certification scenario may be as follows: The quality data may not live up to the expectation of the Certification society, which implies the steel to be re-produced at the steel-mill. Activities that involve steel

production and its quality control are in a SERIAL Block. The failure of the Validation activity necessitates the compensation of the SERIAL Block. Since the SERIAL Block is designed as an atomic activity, the contingency activity for that block can be defined. Contingency activity when defined as REPEAT, forces the activities in the SERIAL Block repeat execution after being individually compensated, hence re-produce steel and test the quality. An example for the use of IF-block in the MARIFlow system is as follows: The certificate numbers are provided in pools to the steel mill by the certification authority. The steal mill consumes the certificate numbers from this pool. Therefore it is necessary to check the certificate number availability in the pool and this is achieved through the IF block as shown in the following FlowDL code segment.

```
START check_pool_numbers () AT szag RETURNS (pool_empty);
IF (pool_empty==true) THEN {
 SERIAL {
  START generate_cert_numbers (OUT cert_numbers)
   AT gl;
  gl SENDS (cert_numbers) TO szag;
  pool_empty=false;
  }
}
ELSE {
}
```

1.4.3.2 Exceptions Raised by the Communication Infrastructure

Various types of errors can be encountered during communication between two agents, or communication with a database system or a mail server. The communication system should recover from any possible failures, retry the operation when possible and inform the user program if the request cannot be issued. The errors can be categorized in the following groups:

1. Connection Errors: The destination machine can reject the request because of any site dependent reason, the server may be down, maximum number of users that can be served simultaneously has been reached etc.

2. Communication Errors: The messages can be lost, or the connection may be too slow for the sender to behave properly. These communication failures can be detected by using time-out mechanism and inserting checksums for the packets that are transferred.

3. Protocol Dependent Errors: Protocol dependent failures occur when the receiver side refuses the continuation of thecommunication because of an abnormal situation, the username /password combination may be invalid for mail server or database server connection, the destination may not be able to accept the request etc.

When an error of any of the categories defined above occurs, the communication system closes the connection and cleans up any temporary resources, like open files and streams. After a certain amount of time the whole operation is retried again (Note that the retry may not be the issue in some protocol dependent errors i.e. in authentication and authorization errors). If the communication fails in all tries, because of any of the reasons stated above, the user program is informed of the failure. The task owning the failed request is aborted, the compensation activity is started and afterwards the workflow is rolled back if there is no contingency activity belonging to that task.

1.4.3.3 Exceptions caused by NON-VITAL Activities

It should be noted that when document flow is a part of a workflow system, more often than not, there will be a need to archive the documents. The transfer and archival of the documents may take considerable amount of time. Therefore a mechanism which allows the other activities in the system (that does not use these documents) to proceed without waiting these archival activities provides for better performance. Yet there should also be mechanisms to guarantee that the workflow instance will not terminate before the successful termination of all such activities. We use NON-VITAL activities suggested in [3] with some modification. Originally the NON-VITAL activities are defined to be those, whose failure does not effect the flow of the process. On the other hand we say that NON-VITAL activities are those that can be assumed to terminate as soon as they start but raise an exception when they fail. In this way a NON-VITAL activity does not delay the execution of other activities unnecessarily; yet their successful termination is guaranteed by the exception handling mechanism. The complete framework for handling NON-VITAL activities within the scope of MARIFlow system architecture is as follows: During the compilation of the process definition, for each NON-VITAL activity, say NVactivity(à.), in order to try that activity a predefined number of times before raising an exception in case of its first failure, the following serial block is generated automatically:

```
SERIAL {
  I=0;
  WHILE (I < NoOfTries OR NVactivity(...)
    terminates unsuccessfully)
  {
    NVactivity(...);
    I=I+1;
  }
}
```

The start guard of this SERIAL block is set to "TRUE" when NVactivity(...) fails for the first time. NoOfTries can be set to a default value or can be obtained from the workflow designer. If this block successfully terminates, it means that NVactivity(à) was successful in one of those trials. However if the SERIAL block fails, a manual user activity is invoked informing the workflow administrator (super user) and the related user that a manual activity needs to be performed for successful termination of this activity

(like sending a document through fax or archiving the document manually). There is one "exception handler" activity automatically generated and placed (within the scope of a SERIAL block in relation to all other activities of the workflow) as the last activity of each workflow instance by the workflow definition compiler. The "terminate successfully" guard expression of this activity for each of the NON-VITAL activities, initially contains "TRUE" values. When the SERIAL block defined above fails (that is, a NON-VITAL activity fails after the predefined number of trials), the guard corresponding to exception handling activity is set to "FALSE" to prevent this workflow instance from terminating before the raised exception is handled. When the manual activity required by the exception is successfully handled, the user through an interface notifies this fact to the system and the guard value "FALSE" is converted to "TRUE". Clearly a workflow instance will not be able to terminate until all the expressions in the exception handler activity become "TRUE" implying that all the raised exceptions are successfully dealt with.

1.5 Conclusions

We describe the design and implementation of a workflow system for document and control flow over the Internet realized through cooperating agents. The architecture provides for the declarative specification and automatic generation of the application rather than producing large amounts of application specific code.

The architecture is general enough to be applied to any domain however the example application is provided for maritime industry and the GUIs are customized accordingly.

The system is developed within the scope of the INCO-DC 97 2496 MARIFlow project and is fully operational for industrial use.

2 Project Objectives

The aim of MARIFlow Project is to provide an architecture for automating and monitoring the flow of control and data over the Internet among different organisations. This "electronic medium", capable of delivering value-added services to the participants, encompasses many different technological areas: from communication to security, databases, transaction support and agents. The project will make use of these technologies to produce a workflow management system. In particular, the goal of the project is to develop an adaptable workflow engine through which the activities of the different participants in the maritime industry can be harmonised, combined, and expanded through better tracking of functional dependencies and documents, improved data access and handling, and lower administrative overheads.

3 Project Achievements

Following targets have been achieved in the MARIFlow project up to now:

Requirement analysis of the application domain finished.
Specifications of the business scenario completed.
A generic architecture for distributed workflow management is designed and partially implemented
MARIFlow cooperating agents (MARCAs), which are capable of executing a workflow instance, are designed and implemented
A workflow definition language that specifies the flow of control and data among MARCAs, is designed and the compiler for this language is implemented
The applications ,which provide interfacing between the agents outside the company domain and the users inside, are designed and partially implemented
A monitoring tool, that is capable to view the status of ongoing workflow instances as well as other process related information is designed and implemented
Authentication and authorization modules, which provide secure communication between agents and secure document transfers, are designed

Local testing of the System

Machines used

Following machines were used in the tests follow, whic are NOT dedicated for the purpose.

beykoz (SUNW,Ultra-5_10 SunOS 5.6,128Megabytes).
beyoglu (Slackware Linux, 128 Megabyte).
altunizade (Red-Hat Linux 128 Megabytes).
haydarpasa (Windows NT 4.0 98 Megabytes).

and beyoglu was used as database server.

Performance Test

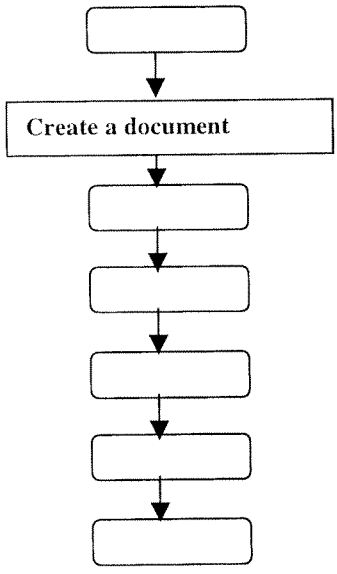


Figure 1 Scenario used during performance test.

A simple scenario that includes a round trip document flow between the participants is tested with several instances started simultaneously and using several document sizes. Document is created at beyoglu followed the path as shown in the Figure 1.

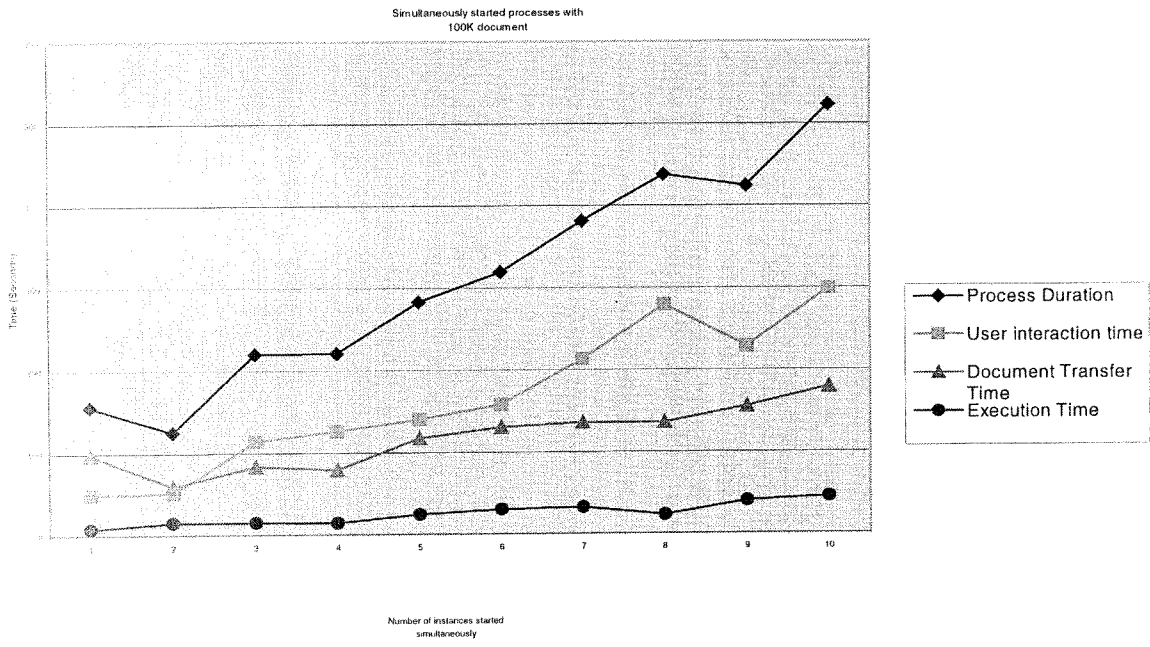


Figure 2 Performance detail for round transfer of 100K sized document with simultaneous executions.

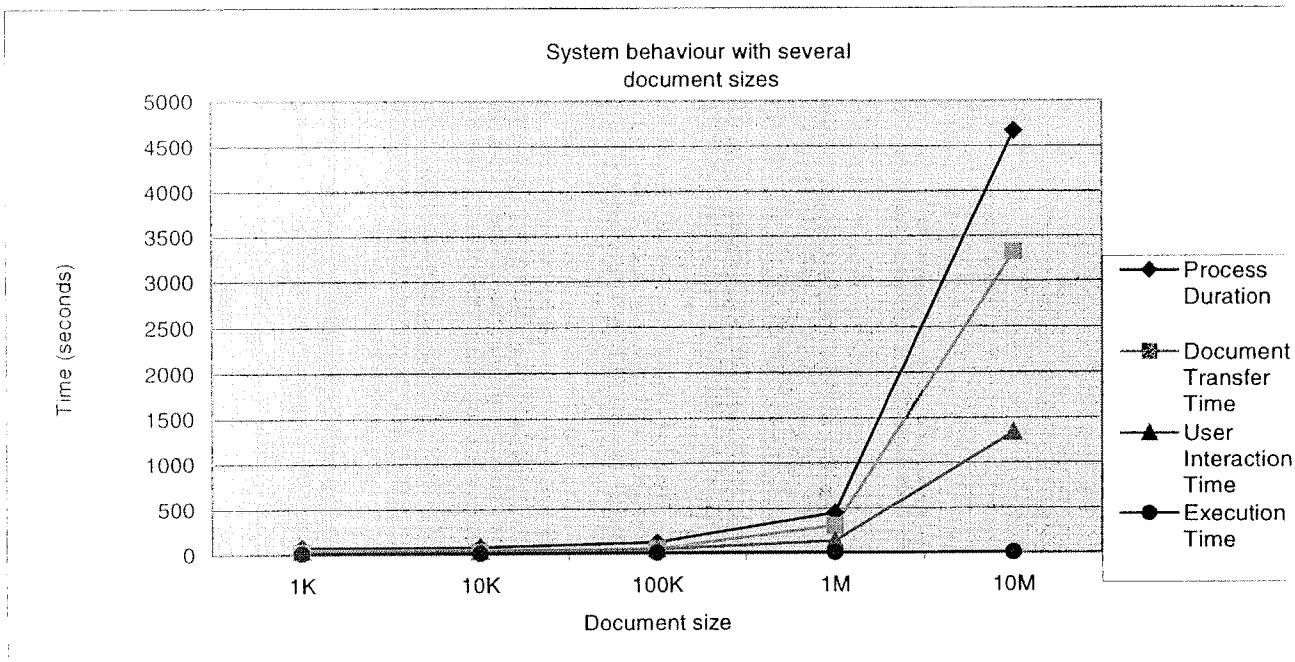


Figure 3 Performance detail for round transfer of a document with several document sizes.

In the first step, several instances of the process were started simultaneously using 100K document. Lifetimes of each activity were recorded. As human was involved in the process, user interaction time for the activity that created the document is also recorded. From the overall process duration, task lifetimes were subtracted to get the execution time. At second step, transfer times for several document sizes were recorded. As can be seen from the figures, process duration and document transfer times increases with the simultaneous instantiations of the process. However, the execution time does not change much. The duration of document transfer time is mainly caused by the communication layer which send documents using 96K sized packages and the sender sends the next package to the receiver when it receives acknowledgement from the receiver that the previous package was written successfully. Taking into account the user interaction in Certification process, this overhead is negligible.

Load Test

In this phase of testing, the certification scenario is tested beyoglu being szag, altunizade as gl, haydarpasa as bal and beykoz as isisan. In the first step, 200 instances of certification process were started at a rate of 12 instance per minute without invoking external applications. All the instances were started successfully. Since our Kernel allocates space in task basis, i.e. memory space is allocated for only living tasks of living instances, system resources are used efficiently.

At second step, 50 instances of the Certification process were started at a rate of one instance per minute with invoking external applications. Since invoked applications involve human interaction, 14 instances were committed in 2 hours and lifetimes of all the activities were recorded as shown in Figure 4. As can be seen from the figure, execution times of activities that sends document(s) are much more less than that activities which invokes external applications inside the firewall user involved in.

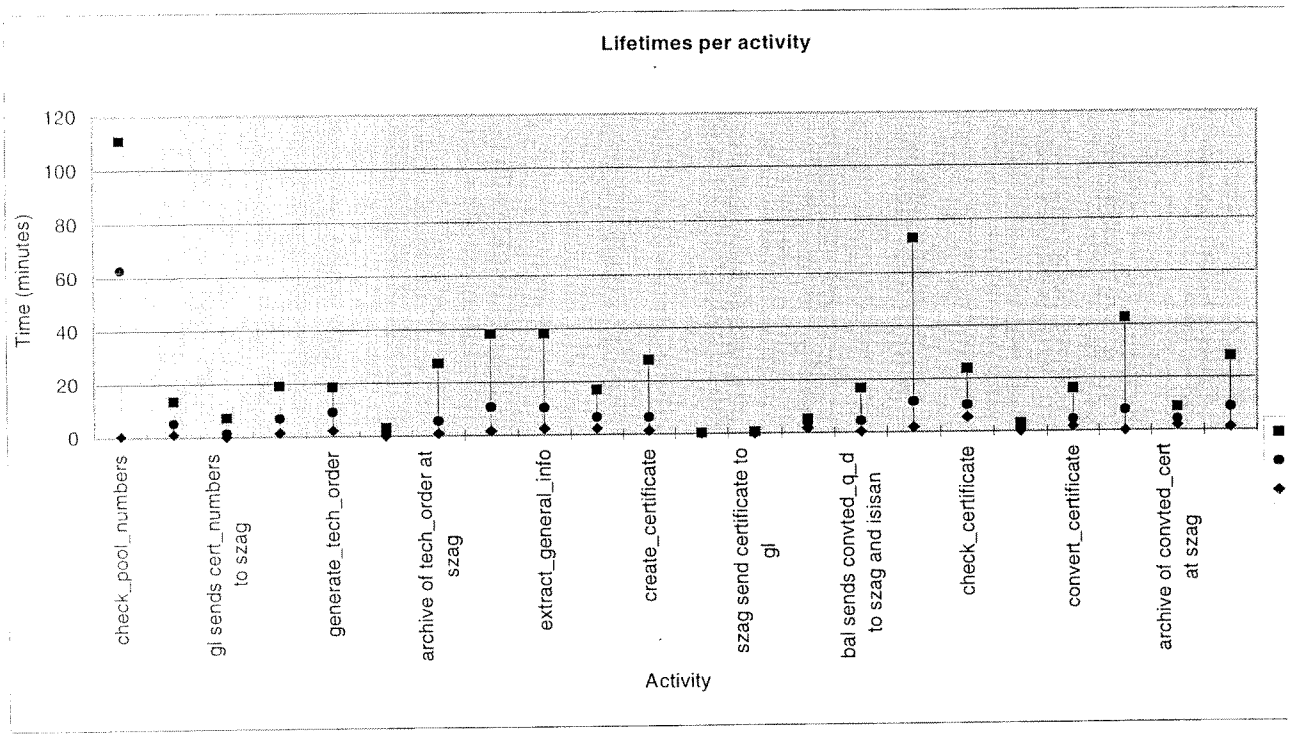


Figure 4 Execution times per activity for Certification process with invoking external applications.

Testing the System over the Internet

A number of tests carried out between 19 May - 7 Jul among BALANCE-METU and BALANCE-GL-METU.

Test on 19 May 2000

Test was carried out between BALANCE and METU. The aim was to initialize components with keys and test a simple process between BALANCE and METU.

Key Initialization Phase :

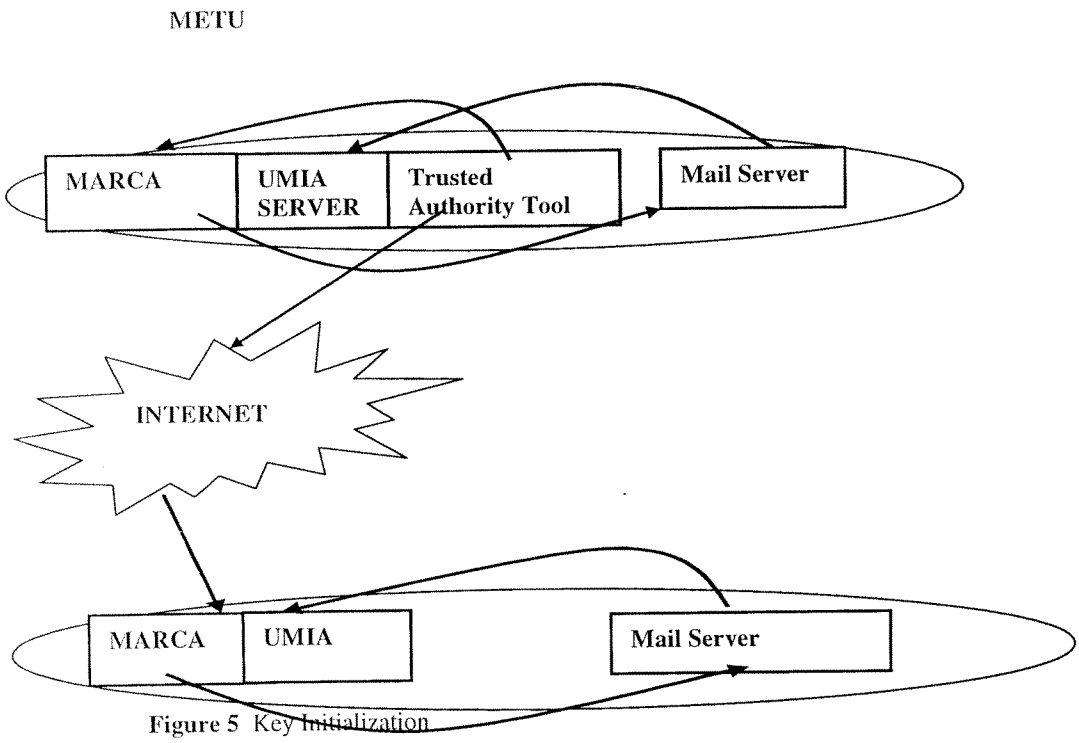


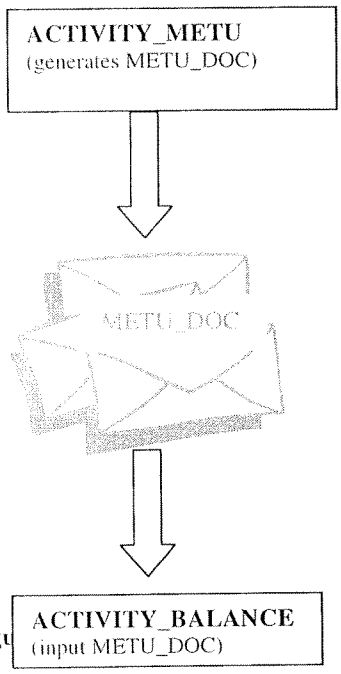
Figure 5 Key Initialization

Key initialization phase was carried out as METU being the Trusted Authority. METU generated key pairs for MARCAs and UMIAs at BALANCE and METU. After each MARCA received necessary keys for itself and its UMIAs, they send keys for UMIAs to the UMIA server by e-mail. Each UMIA server received the mail and initialized with necessary keys.

Process Definition: Transfer a document between METU and BALANCE. Activity_ METU task starts at METU and generates METU_DOC. METU sends METU_DOC to BALANCE. Finally, Activity_BALANCE starts at BALANCE.

Process initialized and an instance is started. Activity_METU task committed from METU after attaching a dummy document to it. After that, MARCA at METU send document

generated by Activity_METU task to BALANCE. After document arrived at BALANCE, Activity_BALANCE task started at BALANCE.



Figu

- **Tests on 24-25 May**

On 24 May, again a simple scenerio tested between METU-BALANCE, but this time the test was unsuccessfull , probably due to a bug at Mail Receiver fixed later. The output of UMIA_Server is below.

Content-Type: APPLICATION/octet-stream; name="/home/mariflow/todorov/deneme/Conn ectionHandler.java"
Event beginning decoding 1
java.lang.NullPointerException at WorklistManagerServer.MailReceiver.decodeAttachment(Compiled Code) at WorklistManagerServer.MailReceiver.parseActivityStartMail(Compiled Co de) at WorklistManagerServer.MailReceiver.checkForMails(Compiled Code) at WorklistManagerServer.MailListenerThread.run(Compiled Code)
!!Error attachment can't be decoded. **********

On 25 May, a sequence of tests are carried out between BALANCE and METU, but some messages couldn't be send from MARCA at METU to MARCA at BALANCE.

Reason: MARCA at METU was running from wrong directory (local tests directory). So, MARCA at BALANCE received the messages from MARCA at METU with message identifiers that were received before, so treated some messages as already received and silently ignored them.

- Test on 31 May

On 31 May, GL participated to the scenario. So, key initialization phase needed to be carried out again METU as being the Trusted Authority. BALANCE,GL and METU received keys correctly.

Key Initialization Phase:

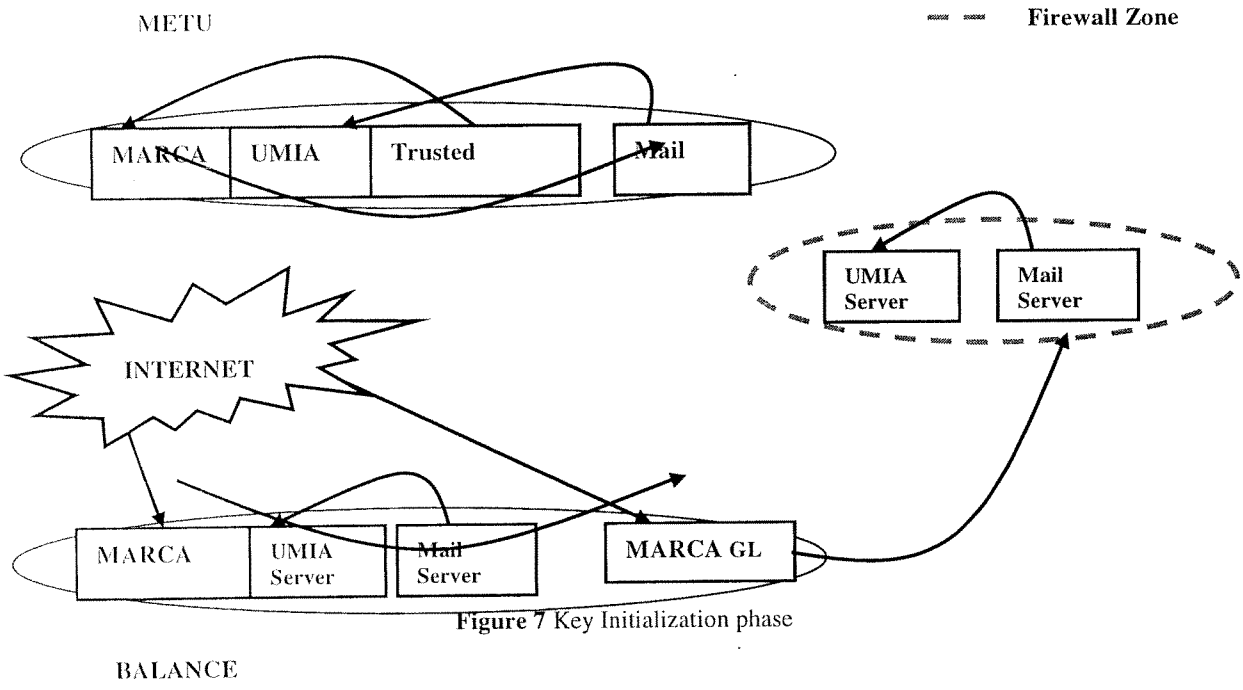
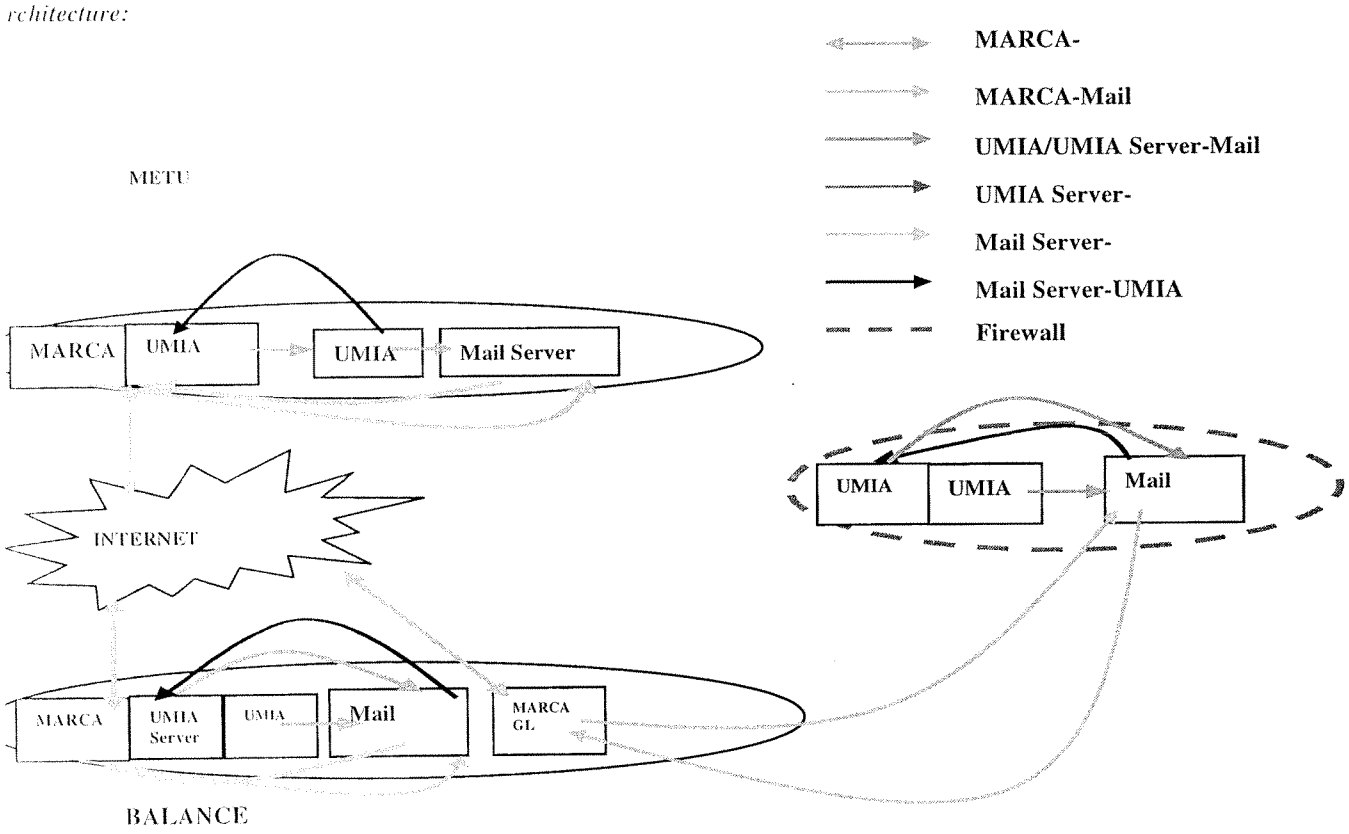


Figure 7 Key Initialization phase

rchitecture:



- Connection between METU-GL was successful, document received by GL MARCA and task successfully started at GL.
- Connection between METU and BALANCE was also successful but UMIA Server at balance was unable to read the mails.

  Reason: A key mail remaining from the old tests was blocking the Mail Receiver.

- **Test on 20 June 2000**

On 20 Jul 2000, two processes involving a document flow from METU-BALANCE-GL and METU-BALANCE-GL-METU are tested. MARCA at BALANCE received documents correctly for both process instances, but neither of them was send to the MARCA at GL.

> **Reason:** Probably due to a configuration error.

- **Test on 6-7 July 2000**

Next round of testing was carried out on 6-7 July, again connection between BALANCE-GL was unsuccessful on 6th of July.

> **Reason:** Alias configuration files of MARCAs GL and BALANCE was wrong. In host addresses section, IP addresses should be used rather than host names since it was configured so in key initialization phase.

While testing connection between METU and BALANCE, UMIA server at BALANCE received the same error faced on May. This time, the bug is found and fixed by METU.

Testing the System with a Different Application

Supply Chain Automation on the Internet through MARIFlow

MARIFlow is a very generic workflow management system that can be used in a wide range of application domains. It has the capability of sending and receiving data and coordinating the executions of the applications that makes MARIFlow appropriate also for domains with high amount of information flow. Independence of each MARCA allows simulating different roles in a domain. It is a complete prototype with security, failure and exception handling components. It has a very powerful workflow definition language and IF block makes it possible to define semantic exceptions.

In order to test and debug the MARIFlow system in a different application domain, we have implemented a supply chain automation scenario. This scenario is influenced by a document [CommerceNet, Catalogs for the Digital Marketplace, Note 97-03, 1997] produced by the CommerceNet consortium. In fact, the most notable work on supply chain automation and integration comes from two main industry consortiums, CommerceNet consortium and the RosettaNet consortium. The CommerceNet is the leading industry association for electronic commerce and in the technical report mentioned they describe the required properties of supply chain integration. RosettaNet project on the other hand, stresses the importance of open content and open transaction standards for supply chain integration and is producing the standard descriptions for computer industry as XML DTDs.

Automation of processes on the supply chain, according to CommerceNet involves the following: Whenever a product is bought, this information should propagate down and up the supply chain automatically triggering a series of distribution, manufacturing and logistics events. As an example, the items collected in a shopping cart should automatically trigger the issuing, approval and delivery of related purchase orders electronically to the appropriate vendor organization. In response an electronic message should be sent to the buyer confirming the acceptance of the transaction, providing tracking number of the transaction and summarizing the status of the order. At the seller site, on the other hand, the related sub processes like shipment and payment need to be automatically activated. Assuming that the buyer is a customer who contacted a retailer, it is necessary to automatically trigger the processes down the supply chain alerting necessary processes in related distributors and manufacturers.

It should be noted that the full automation of customer orders down the supply chain, that is from retailer down to manufacturer, requires the ability for automated stock control in the involved parties. In other words, it is essential that a "stock danger level (SDL)" (which could as well be zero) must be associated with each item in stock and a triggering mechanism must be available raising a signal if "in-stock-quantity" of a product goes below a "stock danger level". This triggering mechanism is needed to automatically enact the related workflows to generate an order to purchase (or produce) the missing item. Since all the current database systems in the market today support a triggering mechanism, this requirement can very easily be satisfied.

Another point is that, in B2B commerce, there is usually a "minimum order quantity (MOQ)" that needs to be satisfied. That is, a party can not go ahead and order any amount it needs to replenish its stock but needs to order a prespecified amount, called MOQ. In the following section we provide a scenario that takes these issues into account.

Description of the Scenario

The supply chain scenario we present contains retailers, distributors and manufacturers. Each retailer, distributor and manufacturer has its own database. In these databases the amount of

goods in the stocks, minimum order quantity, and the total number of remainders (TNR) from the previous unfinished orders are kept. In order to be able to thoroughly test the MARIFlow system we have provided a detailed scenario that involves complicated decision making at several points.

The scenario starts with a customer order. Then the workflow defined for the retailer considers all conditions that can happen by checking the retailer's database. The conditions can be summarized as in the Figure 9.

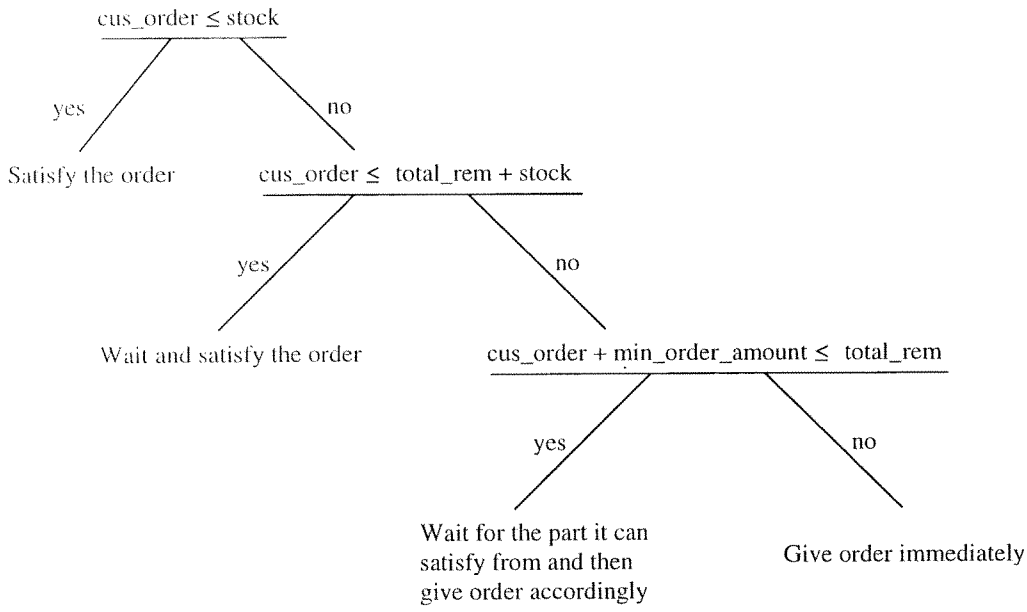


Figure 9 Decision Conditions

I If the customer order is smaller than the stock value of the goods in the retailer, the system will satisfy this order immediately and will modify the database accordingly.

When the amount in customer order is greater than stock but smaller than or equal to the sum of stock value and TNR the system decides on which unfinished order's remainders it should use. It reserves these amounts from the unfinished orders and it waits for these unfinished orders to be satisfied.

If these expected unfinished orders are completed successfully, it will also satisfy the customer order. However if one of the orders that the retailer is waiting for fails to satisfy, then the compensation task of the failed order will start. This compensation task will undo the reservation made by the waiting retailer. Waiting retailer's workflow instance, observing that the orders it waits have aborted, considers the conditions once again to decide what to do to satisfy the customer's request.

If these two conditions are not satisfied, then the retailer workflow must give an order to the distributor to satisfy the customer order. There are also two cases in this situation. The retailer workflow may satisfy whole customer order with the goods it receives with the order sent to the distributor, or it may satisfy a part of the customer order with one of the methods in the first two cases.

This decision is made according to customer order, minimum amount order, stock amount and TNR. First the retailer workflow calculates the shortage when it creates an order to

distributor with maximum available order amount not exceeding customer order. (When the customer order is 230 and minimum order amount is 100, the maximum order that can be created without exceeding 230 is 200. Thus the shortage is 30.) If this shortage can be satisfied with the stock or TNR the procedures in the upper two cases are applied and an order of maximum amount that does not exceed customer order is send to distributor. If this shortage cannot be satisfied with stock or TNR, an order of minimum available amount greater that customer order is sent to distributor. (If 30 can be satisfied from stock, an order of 200 is sent to distributor and 30 is satisfied from stock. If 30 can be satisfied from TNR, the workflow instance at the retailer make reservations to orders and waits till they succeed, and when they succeed creates an order of 200. If both are not possible retailer creates on order of 300.)

When retailer workflow creates an order, flow of control passes to distributor. Sending order is achieved by creating an order file and sending this file to distributor with MARIFlow .File transfers among the parties is explained in Figure 10. When workflow instance at the distributor satisfies the order the flow of control passes back to retailer. Distributor workflow sends to the workflow instance at retailer an invoice file with MARIFlow. (See activities and workflow definition for more information.) If any part of the stock or TNR is reserved, that part is completely used to satisfy customer order.

If we consider how distributor workflow satisfies the order that retailer sends to it, we will see that this process is same as the satisfying customer order process taking place at retailer, only partners change. Here retailer takes the role of customer (creates an order that start up whole process at distributor) distributor takes the role of retailer (tries to satisfy the order requested) manufacturer takes the role of distributor (if distributor workflow somehow can not satisfy the whole order with its own resources an order to manufacturer is created).

The picture is different for manufacturer. When an order is submitted to the workflow instance at manufacturer the only option for manufacturer is to satisfy it with its stock. If this is not possible, the manufacturer workflow should start the production of the goods.

We have considered two cases, which are presented at the Tested Scenarios section.

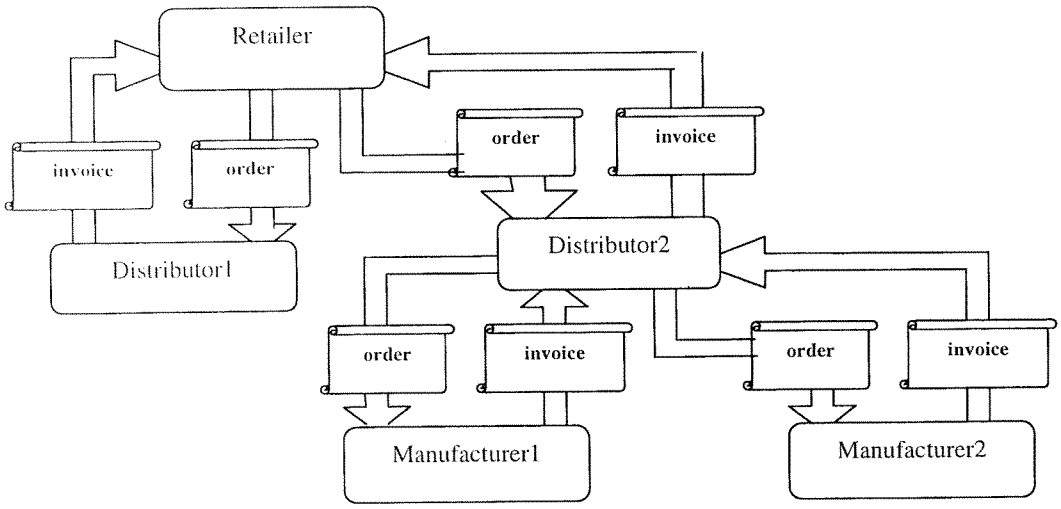


Figure 10 File Transfer among the Parties

The described scenarios are defined in the MARIFlow system. When the retailer workflow wants to make an order to the distributor, it prepares an "order" file and sends it to the distributors. This file contains product id and amount of order. In the same way, if the distributor workflow wants to make an order to manufacturer it also prepare this file in the same format. The second type of the file used in the supply chain scenario is "invoice" file prepared when an order is finished successfully. This file contains the product id, product amount and the cost of the total order.

Implementation Details of the Scenario

Each partner of the scenario has a MARCA installed. A workflow instance is created at the retailer when a customer order arrives. There are one retailer, two distributors, and two manufacturers in the system. The partners and corresponding MARCA installations are as follows:

topkapi.srdc.metu.edu.tr	Retailer
ciragan.srdc.metu.edu.tr	Distributor1
beyoglu.srdc.metu.edu.tr	Distributor2
altunizade.srdc.met.edu.tr	Manufacturer1
haydarpasa.srdc.met.edu.tr	Manufacturer2

The workflow definition used is as follows:

```
PROCESS retailer ();
ACTIVITY retailer_lookup(OUT document order OUT document cus_order);
ACTIVITY distributor_lookup(IN document order OUT document man_order OUT document
invoice);
ACTIVITY manufacturer(IN document man_order OUT document man_invoice);
ACTIVITY retailer_process_order(IN document invoice);
ACTIVITY distributor_process_order(IN document man_invoice IN document order OUT
document invoice);
ACTIVITY retailer_compansate(IN document cus_order);
ACTIVITY distributer_compansate(IN document order);
ACTIVITY create_invoice();

DOMAIN_DEFINITION {
        topkapi.srdc.metu.edu.tr retailer;
        ciragan.srdc.metu.edu.tr distributer1;
        beyoglu.srdc.metu.edu.tr        distributer2;
        altunizade.srdc.metu.edu.tr     manufacturer1;
        haydarpasa.srdc.metu.edu.tr     manufacturer2;
}

VARIABLES {
        boolean found;
}
```

```
COORDINATOR retailer

DEFINE_PROCESS retailer ()
{
 START retailer_lookup (OUT order OUT cus_order) AT retailer
        COMPENSATED BY retailer_compansate (IN cus_order) RETURNS (found);
 IF (found) THEN {
 START create_invoice() AT retailer;
 }
 ELSE {
   retailer SENDS (order) TO distributer1;
   START distributor_lookup (IN order OUT man_order OUT invoice) AT distributer1
RETURNS (found);
   IF (found) THEN {
     distributer1 SENDS (invoice) TO retailer;
   }
   ELSE {
     retailer SENDS (order) TO distributer2;
     START distributor_lookup (IN order OUT man_order OUT invoice) AT distributer2
       COMPENSATED BY distributer_compansate (IN man_order) RETURNS (found);
     IF (found) THEN {
     }
     ELSE {
       distributer2 SENDS (man_order) TO manufacturer1;
       START manufacturer (IN man_order OUT man_invoice) AT manufacturer1 RETURNS
(found);
         IF (found) THEN {
           manufacturer1 SENDS (man_invoice) TO distributer2;
         }
         ELSE {
           distributer2 SENDS (man_order) TO manufacturer2;
           START manufacturer (IN man_order OUT man_invoice) AT manufacturer2
RETURNS (found);
           manufacturer2 SENDS (man_invoice) TO distributer2;
         }
         START distributor_process_order (IN man_invoice IN order OUT invoice) AT
distributer2:
       }
       distributer2 SENDS (invoice) TO retailer;
   }
   START retailer_process_order (IN invoice) AT retailer;
 }
}
```

Applications are written for each activity. MARIFlow coordinate the execution of applications (they are bound to activities) and manages the data flow between applications.

Brief Explanation of Activities

retailer_lookup: This activity executes when a customer order is created. It executes in retailer. Retailer first checks its stock and tries to satisfy the request with its stock. If it cannot, it considers the remainders of unfinished orders that it created. Then it waits and decides in order to satisfy the orders as described above. If the remainders are not sufficient to satisfy the request an order is created immediately. If retailer is able to satisfy the request, it creates an invoice. If the retailer creates an order, this order is transferred to distributors.

distributer_lookup: This activity is very similar to retailer_lookup. Only difference is order id numbers are adjusted. In this activity order requested by retailer is examined and if an order is created it is sent to manufacturer.

manifacturer: This activity is a reduced version of lookup activities described above. It executes on the manufacturer. Manufacturer checks its stock and satisfies the request if possible. The order is aborted if even the second manufacturer fails to satisfy the request.

retailer_process_order: This activity is only executed if retailer creates an order. It receives the order and satisfies the request.

distributer_process_order: This activity is very similar to retailer_process_order. It executes on the distributor. Only difference is an invoice is created as a result.

retailer_compansate: This activity is a compensation activity that executes if an order chain aborts. It adjusts the database such that the order created by retailer is rolled back as if it has not been created.

distributer_compensate: This activity is same as the retailer_compansate. It executes on distributor.

Applications Bound to Activities

The applications are coded in Java. The input to the system is the order file. An order file or/and an invoice file is created according to the type activity. Stock values and order records are stored on the postgresql database.

An Alternative Implementation of Supply Chain Scenario

In the scenario described so far, there is an only minimum order constraint. In real life the retailers also keep a minimum number of goods in their stocks to satisfy the order of the customer as soon as possible. This minimum amount is called Stock Danger Level (SDL). We decided to add this constraint to the Supply Chain Scenario described above. Whenever the number of goods in the stock of either distributor or retailer is below the SDL ,automatically an order is given by the workflow instance to increase the stock value above SDL. The new structure of the scenario is given in Figure 11.

After a customer order arrives to the retailer workflow , considering the database values it decides what to do next. The actions that can be done are as follows:

As in Figure 9 the first condition "satisfying the order immediately". However, the difference is that after satisfying the demand of the customer it checks its stock whether the number of goods is below SDL. If so a new order to the distributor is created to increase the stock

value above SDL. Likewise, in the second condition (i.e., customer order < TNR+stock value) and the third condition (i.e., (customer order%minimum order number) < TNR+stock value) after satisfying the demand of the customer it checks the stock value in the database and decides whether to create an order to the distributor or not. However, in the fourth condition since the retailer satisfies the demand of the customer totally from the order it gave to the distributor, it does not need to consider the SDL (The stock value is not changed). If any of the orders given in order to increase the stock value to stock danger level fails, the party given the order takes an alert message.

We have considered two cases, which are presented at the Tested Scenarios section.

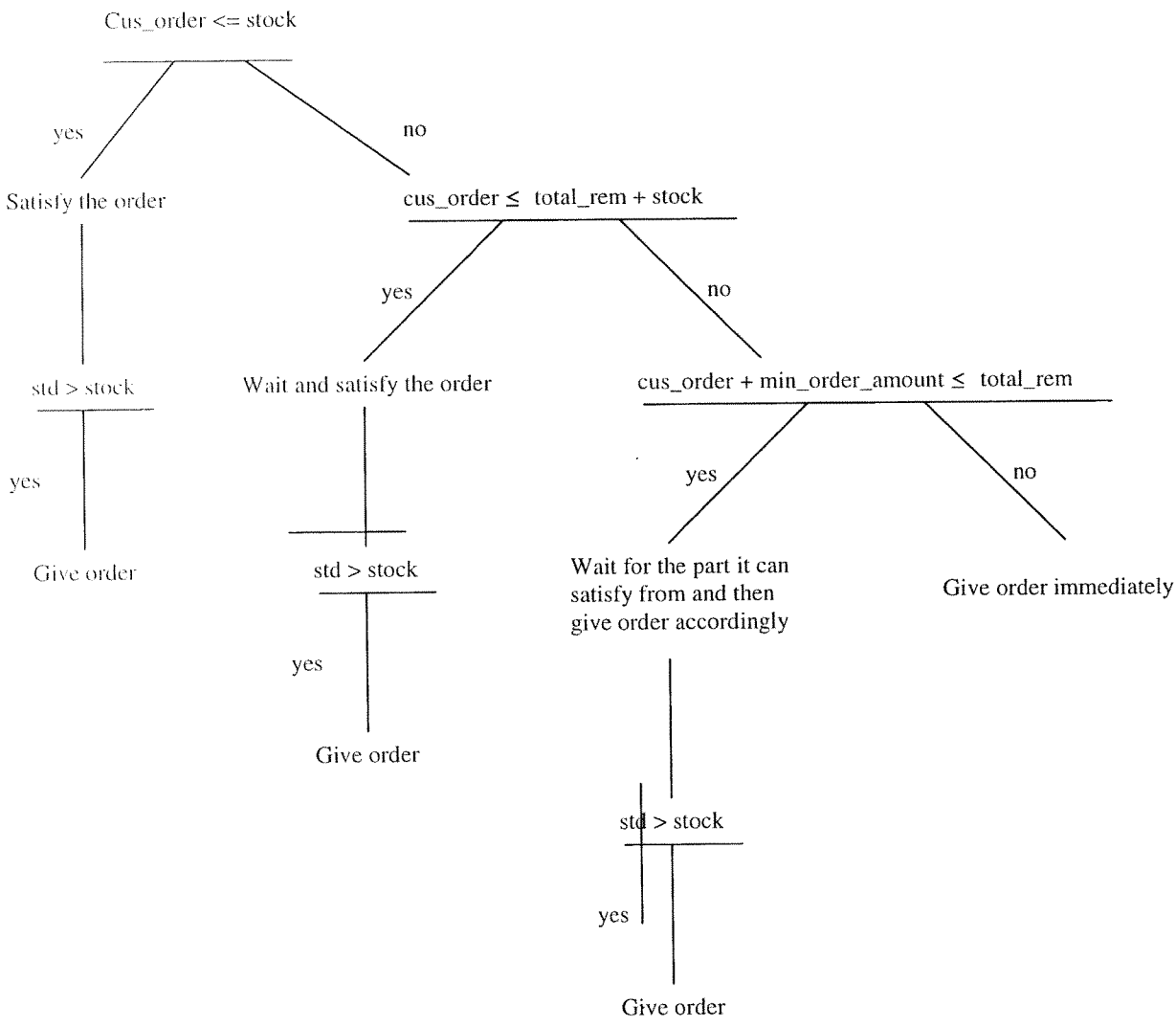


Figure 11 Decision Conditions considering SDL

Scenarios Tested

Four scenarios have been tested. In the first two scenarios the Stock Danger Level is not considered.

i) Wait For Dependencies Scenario (WFDS)

In this scenario, 3 customer order take place. These are 50, 90, 90 orders by customers. The databases at the beginning contain:

retailer	distributor1	Distributor2	manufacturer1	manufacturer2
30	100	0	200	0

When the first customer comes and makes an customer order of 50, the retailer checks its database to see whether it can satisfy the order. It sees that it has 30 of that product, so it cannot satisfy the order immediately. There is also no total_remainder at the beginning, so it can also notsatisfy from there. As a result, retailer makes an order to the distributor1 of 100. Distributor1 also checks its database to see whether it can satisfy this 100 order made by retailer. Figure13 explains the actions taken in this customer order.

Stock	Remainder1	Total_remainder
30	100-50=50	50

Before the 1st customer's order is satisfied by retailer, 2nd customer comes. It orders an amount of 90. The retailer sees that it should make an order of 100 to satisfy 2nd customer's order. Because it can satisfy the order of 2nd customer neither by stock=30 only, nor (stock+total_remainder)=30+(100-50), the 2nd customer only takes from the order retailer has given to distributors which is 100. Figure 14 explains the actions taken in this customer order.

After 2nd customer's order :

Stock	Remainder1	Remainder2	Total_remainder
30	50	30	50+30=80

Before the 1st and 2nd customers' order are satisfiedby retailer, 3rd customer comes. 3rd customer wants 90 of that product. So retailer checks its database and sees that if 30 from stock 50 from remainder1 and 10 from remainder2 is given to customer3, it(retalier itself) needn't make an order to the distributor. So the 3rd customer's order is dependent to stock by 30, to remainder1 by 50 and to remainder2 by 10. So the retailer waits for the orders it has placed to the distributors to be finished to also satisfy the 3rd customer's order. Figure 12 explains the actions taken in this customer order.

dependency list of 3rd customer's order:

Stock	Remainder1	Remainder2
30	50	10

After 1st order of retailer it has done to the distributors is satisfied, 1st customer buys and the buy operation of 1st customer is finished successfully. Now 3rd customer only waits for the second order of retailer. After this second order of retailer is finished successfully (i.e., the second order the retailer is satisfied by any of the distributors), the 2nd customer's buy operation also finishes successfully. Meanwhile, also the 3rd customer was waiting for these two orders of retailer, because it is dependent to these orders' remainders. Because these orders are satisfied by distributors, the3rd customer's order is also satisfied by the retailer. This scenario finishes here.

ii) Abort Scenario

We assume the process starts from beginning, the databases contain:

Retailer	Distributor1	Distributor2	Manufacturer1	Manufacturer2
30	100	0	100	0

Here 3 customer orders are seen again. When the first customer comes and makes an 50 order to the retailer, the retailer after checking its database sees that it should make an order of 100 to the distributors. Figure 13 shows the actions taken in this customer order. After the 1st order which retailer has made to the distributors, the database of retailer contains

Stock	Remainder1	Total_remainder
30	100-50=50	50

Before 1st order of retailer is satisfied(1st customer is still waiting), 2nd customer comes and orders 190. Figure 16 shows the actions taken in this customer order. Retailer sees it should make an order of 200. So, before 2nd order of retailer is not satisfied by distributors or manufacturers, the stock and remainders are :

Stock	Remainder1	Remainder2	Total_remainder
30	50	200-190=10	50+10=60

The 3rd customer then comes and makes an order of 90. Figure 15 shows the actions taken in this customer order. The retailer sees that stock+total_remainder(remainder1+remainder2)=30+50+10 is enough for satisfying this customer order. But for the 3rd customer's order to be satisfied, order1 and order2 of retailer should finish. (Retailer ordered to the distributors).

dependency list of customer order 3:

Stock	Remainder1	Remainder2
30	50	10

iii) We assume the process starts from beginning, the databases contain:

Retailer	Distributor1	Distributor2	Manufacturer1	Manufacturer2
80	150	0	300	400

The minimum order value is 100, and the stock danger level is 70 for both distributors and retailer.

When a customer comes and makes an 90 order to the retailer, the retailer after checking its database sees that it should make an order of 100 to the distributors. After the 1st order which retailer has made to the distributors, the database of retailer contains

Stock	Remainder1	Total_remainder
80	100-90=10	10

Then the order of the retailer comes to the distributor, it sees that it has 150 goods which is enough to satisfy the retailer's demand. It gives it to the retailer making the necessary changes in the database. After this operation the database of the retailer contains

Stock	Total_remainder
90	0

And the database of the distributor1 contains

Stock
50

After satisfying the order of retailer distributer1 checks its stock whether it is below the SDL which is 70.The number of goods in its stock is 50 which is below SDL. So it decides to give an order of 100 to the Manufacturer to increase its stock value.

After getting the order of the distributor, manufacturer1 checks its stock value to see whether it is enough to satisfy the order. It has 300 goods so it satisfies the demand of the distributor, after this action, the databases of the distributer1 and the manufacturer is as follows:

Distributor Stock	Manufacturer1 Stock
150	200

Since all of the stock values of the retailer, distributors and manufacturers are consistent with the SDL, and no new customer order occurs, this test scenario ends here.

iv) In this test scenario the databases contain:

Retailer	Distributor1	Distributor2	Manufacturer1	Manufacturer2
80	250	350	100	100

The minimum order value is 100, and the stock danger level is 70 for distributor1 and retailer, 250 for distibutor2 and 80 for manufacturers.

When a customer makes an order of 350 to the retailer, the retailer checks its database and decides to make an order of 300 to the distributors. The database of the retailer after giving the order is as follows.

Stock	Remainder	Total_remainder
80	350-300=50	50

The order first goes to the distributor1 and since it has not sufficient amount of goods to satisfy the order, the order goes to distributer2. The distributer2 has enough goods and satisfies the demand of the customer. After satisfying the order, distributer2's stock contains 50 goods that is below stock danger level. Hence, it decides to make an order of 200 to manufacturers in order to increase its stock value to its stock danger level. However, neither of the manufacturer has 200 goods in its stock so the distributor2 takes alert message.

After taking the goods from the distributer2, the retailer satisfies the order of the customer and the amount of goods in its stock become 30 as shown below.

Stock	Remainder	Total_remainder
30	0	0

Since the number of goods is below the stock danger level of 70, the retailer makes an order of 100 to distributors. The order goes to the distributer1 and it has sufficient number of goods to satisfy the demand of the retailer. After satisfying the order the databases are as follows.

Retailer	Distributor1	Distributor2	Manufacturer1	Manufacturer2
130	150	50	100	100

At the end of the scenario retailer, distributor1 and the manufacturers are in equilibrium but distributor2 takes alert messages.

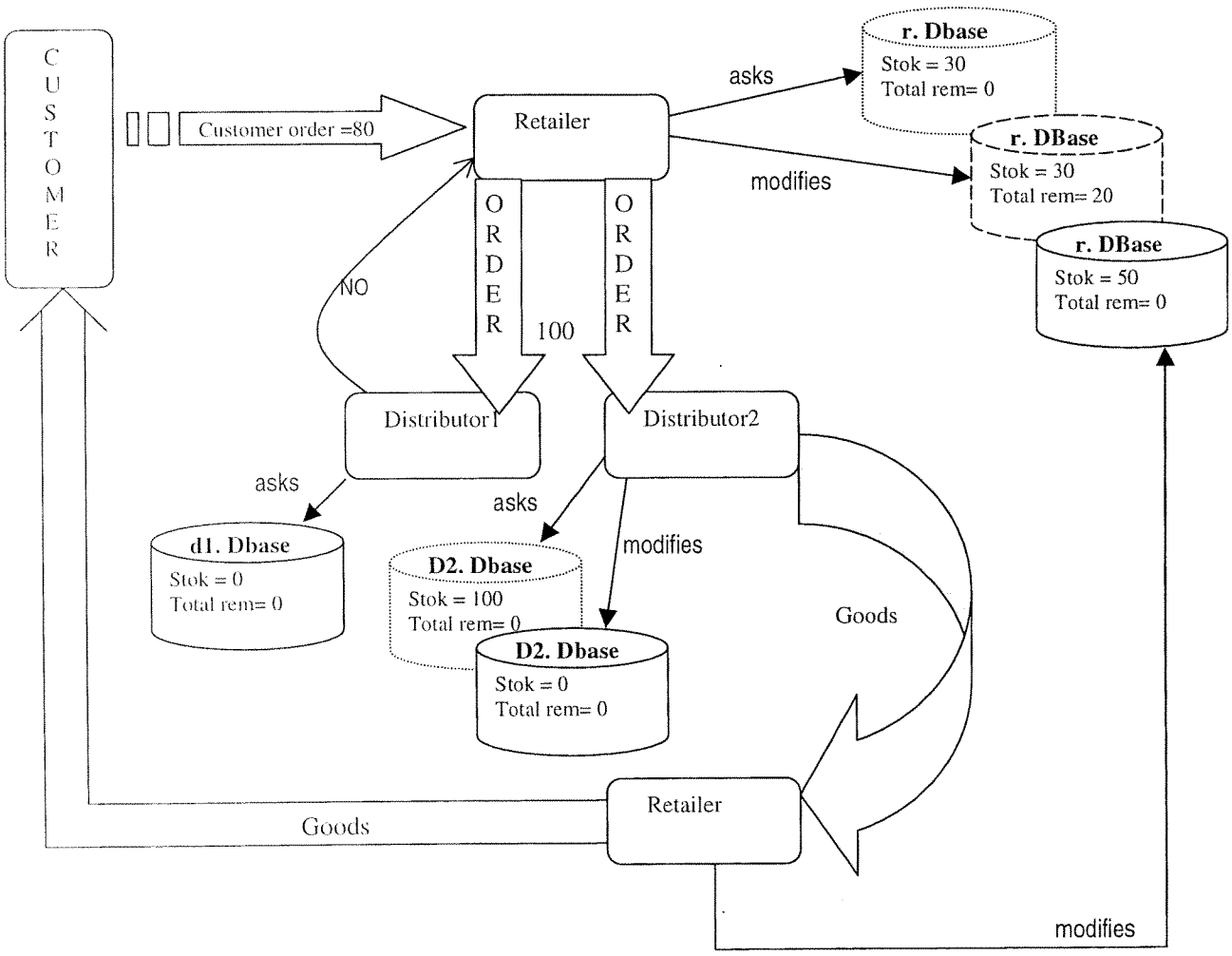


Figure 12 Description of the actions taken for the third customer order in WFDS

But the second order of retailer can not be satisfied, because neither any of the distributors have it, nor the manufacturers(when distributors don't have enough product of any type, they make order to the manufacturers, but here in this situation , also neither of the manufacturers have 200 of the product type). So as a result , the 2nd order of retailer aborts. The sell operation of retailer to the 2nd customer finishes unsuccessfully. Also, because the 3rd customer order also depends on the remainders of the 2nd retailer order which aborted , the operation of selling this product type to the 3rd customer by the retailer also fails. Only the 1st customer's order is finished successfully. (Retailer has find enough amount of that product for 1st customer).

The 2nd retailer order was done in fact to satisfy the 2nd customer's order. But when 3rd customer came, it also waited for this second order. But it failed. But still, there may be a chance for customer 3. The retailer may satisfy this order by making an order to the distributors, in fact in this situation it can satisfy. Because still the databases contain:

Retailer	Distributor1	Distributor2	Manufacturer1	Manufacturer2
30	0	0	100	0

and the remainders are :

Stock	Remainder1	Remainder2
30	100-50=50	ABORTED

when the retailer reconsiders the situation for 3rd customer, it sees that for this 90 order of 3rd customer, neither the stock=50, nor the sum of stock and the total_remainder is enough(30+50=80). So it sees that it should make an order of 100 to the distributors. This is satisfied by the manufacturer 1 (in fact by distributor2, because when 2nd distributor sees that it does not have 100 of that product, it makes an order to the manufacturer1 which is satisfied by this manufacturer) .Then the scenario continues, new customers come and etc...

CONCLUSION

With MARIFlow we were able to simulate the members of a supply chain scenario. MARIFlow was not only a tool for sending and receiving documents, but also a way to coordinate the flow of execution. Thus the flow of orders is completely controlled with MARIFlow.

We not only simulated a simple order-confirm-invoice scenario but also some complex scenarios that involve one instance to wait another or one instance to reconsider the situation when another aborts. Without MARIFlow it would be much more challenging to provide these features in portable modules. MARIFlow, with its underlying workflow architecture provided us a very easy way to achieve these features.

It's easy to see that MARIFlow, with the features it provide and with the independent MARCA architecture is very appropriate to this domain and domains like this.

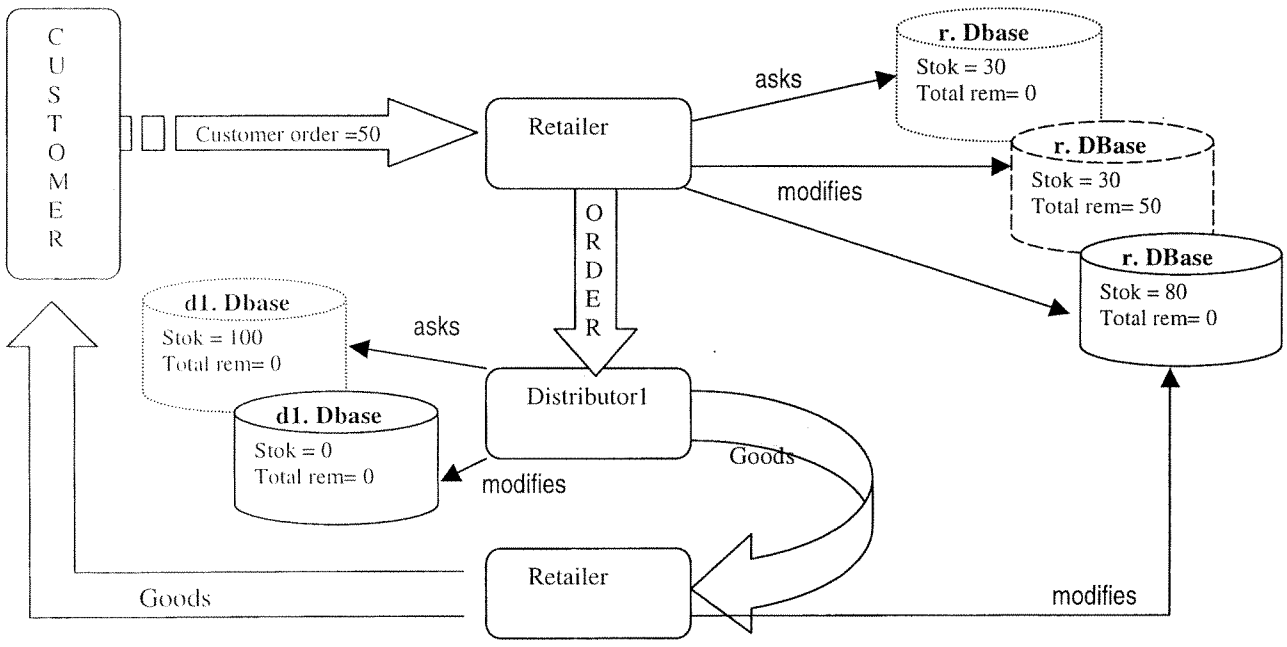


Figure 13 Description of the actions taken for the first customer order for both scenarios

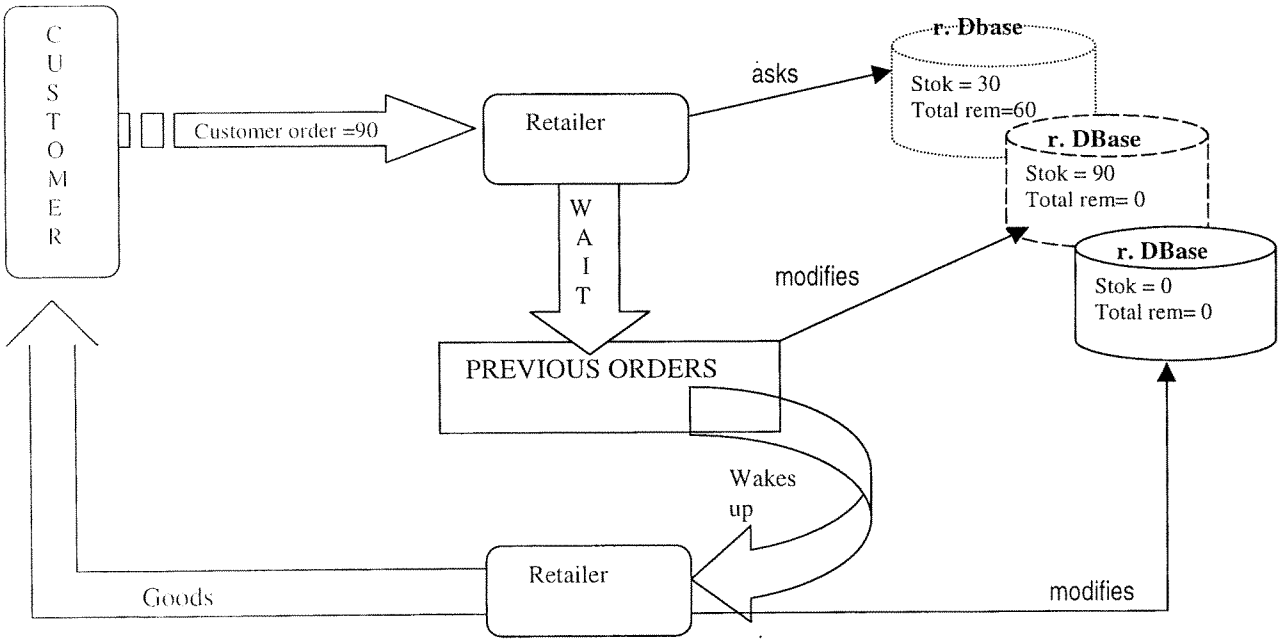


Figure 14 Description of the actions taken for the second customer order in WFDS

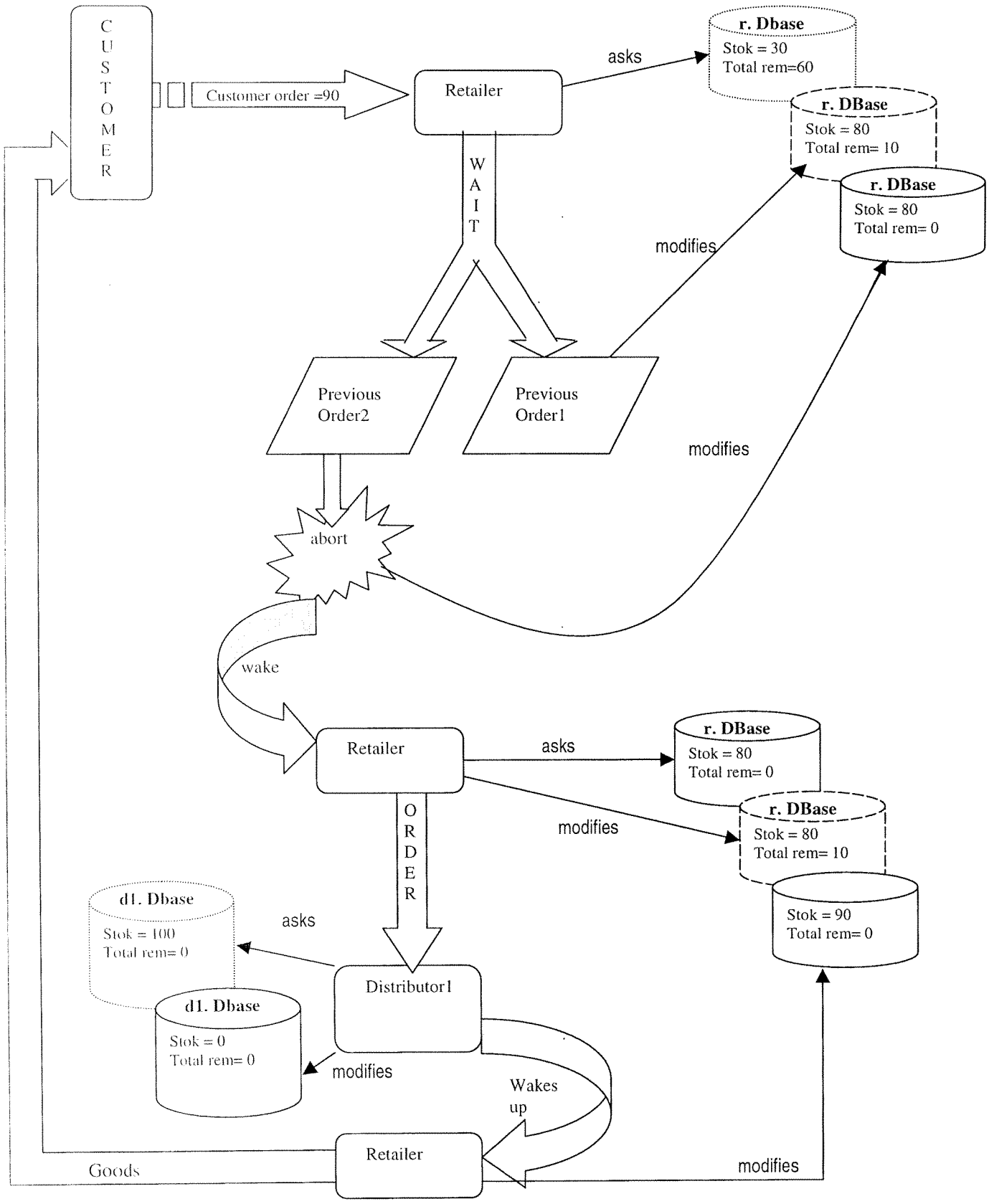

CUSTOMER
Customer order =90
Retailer
asks
r. Dbase
Stok = 30
Total rem=60
r. DBase
Stok = 80
Total rem= 10
r. DBase
Stok = 80
Total rem= 0
WAIT
modifies
Previous Order2
Previous Order1
modifies
abort
wake
Retailer
asks
r. DBase
Stok = 80
Total rem= 0
r. DBase
Stok = 80
Total rem= 10
Stok = 90
Total rem= 0
modifies
ORDER
d1. Dbase
Stok = 100
Total rem= 0
asks
Distributor1
d1. Dbase
Stok = 0
Total rem= 0
modifies
Wakes up
Retailer
Goods
modifies

Figure 15 Descriptions of the actions taken for the third customer order in Abort Scenario

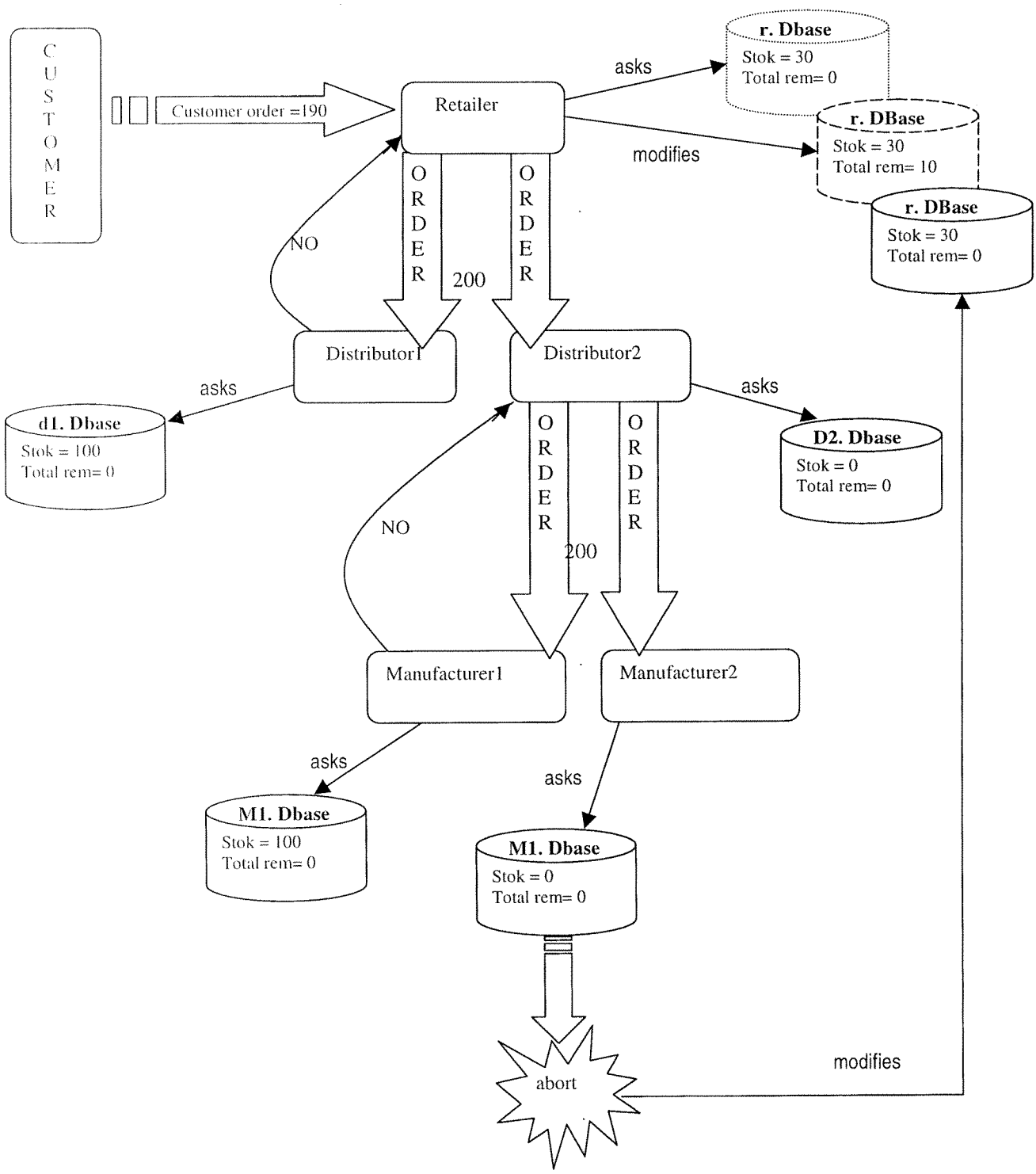


Figure 16 Descriptions of the actions taken for the second customer order in Abort Scenario

4 Availability and scalability issues of MARCAs have been considered and mechanisms for restarting 'dead' MARCAs have been proposed

In Mariflow, for each MARCA installed there is a background process at that site, called the "rescue process". The rescue process is responsible for monitoring the lifetime of the agent and checks the MARCA at specific time intervals through a predetermined socket. A thread of the MARCA listens to this socket and responds to the signals. If the MARCA does not respond to this process for a given period of time, the process starts sending signals more frequently. If the MARCA still does not respond, after sending a bunch of signals the process assumes that the MARCA is not functional. The two possibilities in this case are the MARCA could be blocked or it could be dead. When the rescue process is unable to find the OS process that belongs to this MARCA (i.e., it is dead), it instantiates a new MARCA by the help of the persistent logs related with the state of the agent.

Otherwise if the MARCA is blocked, it is necessary to kill the old instance prior to installation of a new instance. Since the logs are persistent it is possible to recover the state of the MARCA killed and hence the site does not suffer from any inconsistencies.

For the described mechanism to work correctly it is necessary to make sure that rescue process stays alive. Therefore, just as the rescue process checks to see that the MARCA stays alive, the MARCA also checks to ensure that the rescue process stays alive by signaling the rescue process at predefined time intervals. It is MARCA who reinstantiates the rescue process when it dies.

5 Contacts Established with Other Projects

ENHANCE - Enhanced aeronautical concurrent engineering

Contact: Jaques Spee (M.I.S. Organisatie-ingenieurs b.v.), jaques.spee@mis.nl

This ESPRIT project is developing a demonstrator for workflow of engineering data. STEP Application Protocol ISO 10303-233 has been chosen for a standardized description of data. It is specific to engineering data to have a version history enabling tracing of changes. The tool will mainly support a predefined scenario in which the number of participants for a certain role may vary.
System architecture:

- user interface layer
- communication layer (CORBA)
- server layer

Practically, the system shall link a number local workflows. In order to achieve this link, it is essential to have all workflows described in standardized description language. For this purpose, an XML binding of the WFMC model is used. Working in a decentralized mode, there is no central database holding WF information. No workflow co-ordinator or manager is required, once the consortium has been set up. They claim to support dynamic modifications of the workflow definition. The status of an individual activity can characterized as *blocked, enabled, in process, published, committed.*

According to their security concept, encapsulation of information isn't supported, i.e. all data and activities are exposed to every participant. The usage of corporate firewalls has not been considered yet.

MARVIN - Maritime Virtual Enterprise Network (ESPRIT EP29049)

Contact: Clemens Odendahl (IWI, Uni. Saarbrücken),: Odendahl@iwi.uni-sb.de

The project is developing an agent based system called *Maritime Enterprise Integration Tool* (MEIT) to facilitate and co-ordinate the interaction and co-operation of companies building up a virtual organization to carry out the mission of repairing a ship in the shortest time possible.

> System architecture:
> communication layer (Agent2Agent, Agent2User, Agent2System; using WWW, Java)
> networking layer (TCP/IP, RMI)
> processing layer (database management component, reasoning component)

In the beginning, a workflow is modeled as *extended Event Process Chain* (eEPC) using the ARIS toolkit. Based to this model, program code is generated for a number of agents automatically (?). Each agent has to fulfil a predefined role, the (role specific) knowledge based process logic is stored using the KQML language. Creation, configuration and maintenance of agent specific knowledge bases is mainly a manual process (!). The reasoning component, which is also part of an agent, is implemented in Java Expert System Shell (JESS) and CLIPS. An object-oriented system called *Jeevan* is used for database management. For role-specific communication between agents they temporary grant access to their database for specific activities. A specialized agent (administrator agent) is responsible for registration of new users and provision of agent templates. The database of this agent can be considered as database of the workflow engine. Currently there is no monitoring tool yet.

Corporate firewalls haven't been considered since all agents running on the same server. Furthermore, for upload and download of data dedicated port numbers are used. Access is controlled by IP address numbers.

The complete service shall be provided in a single place by, e.g., Web Service Provider, no local installations will be required at client site.

SEANET The product data interchange for the maritime information society (ESPRIT)

Contact: Tim Turner (Lloyds Register), Tim.Turner@lr.org

The objective of this project is to create the virtual-enterprise-wide, internet-based product data management (PDM) system that will enable all maritime enterprises to take full advantage of STEP standardization.

A system is developed interconnecting a number of different PDM systems in order to support seamless exchange of data between those systems. A basic requirement for this exchange is that all participating systems have interfaces to both STEP P21 and XML files.

The flow of activities is modelled in IDEF-0 and converted to a XML representation based the WFMC standard using a system component called GLOBE. This model is then deployed on a special web server, having many functions of a workflow engine like execution of activities, sending messages and acknowledgements. It also supports monitoring. Each partner in the scenario has to operate its own PDM server. The architecture doesn't support corporate firewalls, therefore all servers have to run in public areas. Encryption is an optional feature for data transfer.

Every participant has copy of the global workflow definition tailored to his local perspective. Currently the project partners are in an effort to establish contacts with other related projects.

6 Results and Activities related to Standardisation

To guarantee that the system conforms to existing standards, the international EDIFACT standard (Electronic Data Interchange for Administration, Commerce and Transport) will be used for the exchange of information within the different components in the system and among different participants. The EDIFACT syntax is defined in ISO standard 9735 and parts of the UN Trade Data Elements Directory (UNTDED), which is the basis for EDIFACT data elements are standardized is ISO 7372. At the present project status, it can not be finally decided which of the required data exchange processes will be performed using EDIFACT messages. But all messages (ORDERS, ORDRSP, DESADV, QALITY and INVOIC) needed for the MARIFLOW application areas are still standardized (Status 2: finally released). After deciding on the final project scenario the needed messages will be implemented.
EDIFACT messages are still defined for the exchange of structured information between application systems. But in MARIFLOW there is also a need for structured information exchange between users. Therefor the Extensible Markup Language (XML) which is a subset of SGML (Standard Generalized Markup Language – ISO 8879) will be used. XML has been defined by W3C (World Wide Web Consortium). The recommendations will be applied for the exchange and presentation of structured information based on EDIFACT message contents. Therefore DTDs (Document Type Definitions) have to be defined and tested within the project. The MARIFLOW consortium considers the XML/EDI group and their important documents like e.g. the "Proposal for a UN Repository for XML/EDI", which has been proposed to CEFACT (Center for the Facilitation of Procedures and Practices in Administration, Commerce and Transport)
There are several types of inspection documents specified by the European standard EN 10204 [EN 10204]. The document types that are used in this business scenario are document types 3.1 and 3.2, called Inspection Certificates (3.1) and Inspection Reports (3.2), respectively.
Tests are performed on the basis of national and international standards like EN 10028, DIN, ASME, British Standard or ASTM. Each class society performs the tests according to any of these standards.

7 Other Benefits

The MARIFlow Project brings following benefits:

The architecture provides for the declarative specification and automatic generation of the application rather than producing large amount of application specific code by the help of a block structured nature of the workflow definition language

The system that is being developed attempts to bring solutions to the user interaction through the Java based WEB interfaces, reliable and automated document and data transfers through a distributed agent based architecture and it is more resistant to localized failures than that of centralized systems.

The block-structured nature of the language avoids unreachable states in the workflow execution and also the deadlocks.

The system proposes and adaptable design and implementation so that the engine will run in different platforms ranging from a small set of personal computers to high end workstations and mainframes by the universal programming language Java.

The architecture is general enough to be applied to any domain other than the maritime domain. The system, workflow specification, communication details and database activities are suitable for any kind of domain definition.

8 Information Dissemination Activities

Here is the list of information dissemination activities, which are carried out up to now:

The project is presented at the "International Maritime forum" in Kavala, Greece on May 21st, 1999.

MARIFlow project is presented at the WONDERMAR workshop on May 20th, 1999 in Kavala, Greece.

MARIFlow project is presented and the prototype is demonstrated at the WONDERMAR workshop on December 7th, 2000 in Bremen, Germany.

The following publications describe the MARIFlow project:

Dogac, A., Tambag, Y., Tumer, A., Ezbiderli, M., Hamali, N., "An Agent-based Workflow System for Inter-enterprise Business Processes", in Proc. of European Commission's eBusiness and eWork Conference and Exhibition, Madrid, Spain, October 2000.

Dogac, A., Tambag, Y., Tumer, A., Ezbiderli, M., Tatbul, N., Hamali, N., Icdem, C., Beeri, C., "A Workflow System through Cooperating Agents for Control and Document Flow over the Internet", in Proc. Of the Intl. Conf. on Cooperative Information Systems (CoopIS '00), Israel, September 2000.

Dogac, A., Tambag, Y., Tumer, A., Ezbiderli, M., Hamali, N., "An Agent-based Workflow System for Inter-enterprise Business Processes", in Proc. of European Commission's eBusiness and eWork Conference and Exhibition, Madrid, Spain, October 2000.

U. Langbecker, M. Lehne, G. Alonso; "How to support material certification as virtual business process", Proceedings of the International Conference *e-business for Industry*, London, June 20-22, 2000

U. Langbecker; "How to support material certification as virtual business process", *3rd WONDER MAR Public Workshop*, Copenhagen, May 26, 2000

U. Langbecker; "Möglichkeiten zur elektronischen Unterstützung der Materialzertifizierung" (in German), *workshop of the Association of German Mechanical Engineers (VDMA)*, Hamburg, July 4, 2000

M. Lehne, "quality certificates through co-operating agents via internet , an inter company workflow service for the maritime industry", *Conference daratechMarine2000 , January 24-26, 2000 in Houston Texas*

Dogac A., Ezbiderli M., Tambag Y., Icdem C., Tumer A., Tatbul N., Hamali N., Beeri C. "The MARIFlow Workflow Management System", in "16th International Conference on Data Engineering" in San Diego, USA on March 1st, 2000.

Dogac, A., Beeri, C., Ezbiderli, M., Tatbul, M., Icdem, C., Erus, G., Cetinkaya, Dogac, A., Ezbiderli M., Tatbul, N., Icdem, C., Erus, G., Cetinkaya, O., Tumer, A., Beeri, C., "A Workflow System through Cooperating Agents for Document Flow over the Internet", Technical Report.

Cingil, I., Dogac, A., Tatbul, N., Arpinar, S., "An Adaptable Workflow System Architecture on the Internet for Electronic Commerce Applications", In Proc. of Intl. Symposium on Distributed Object Applications, Edinburgh, September 1999.

Hagen, C., Alonso, G., "Highly available Process Support Systems implementing backup mechanisms", 18th IEEE Symp on Reliable Distributed Systems (SRDS'99), Lausanne, Switzerland, October, 1999.

Schuldt, H., Popovici, A., Schek, H.-J., "Give me all I pay for – Execution Guarantees in Electronic Commerce Payment Processes" , In Proc. of the Informatik'99 Workshop "Unternehmensweite und unternehmens-übergreifende Workflows: Konzepte, Systeme, Anwendungen", Paderborn, Germany, October 1999.

Dogac, A., Beeri, C., Ezbiderli, M., Tatbul, M., Icdem, C., Erus, G., Cetinkaya, O., Hamali, N., "MARIFlow: A Workflow Management System for Maritime Industry" , In Proc. of 'Application of Information Technologies to the Maritime Industry', Eds. C.Guedes Soares and J.Brodda, MAREXPO Consortium, 1999, pp. 33-51.

Dogac, A., Ezbiderli M., Tatbul, N., Icdem, C., Erus, G., Cetinkaya, O., Tumer, A., Beeri, C., "A Workflow System through Cooperating Agents for Document Flow over the Internet", Technical Report.

Arpinar, B., Halici, U., Arpinar, S., Dogac, A., "Formalization of Workflows and Correctness Issues in the Presence of Concurrency", Journal of Distributed and Parallel Databases, Vol. 7, No. 2, April 1999, pp. 199-248.

Cingil, I., Dogac, A., Tatbul, N., Arpinar, S., "An Adaptable Workflow System Architecture on the Internet for Electronic Commerce Applications", In Proc. of Intl. Symposium on Distributed Object Applications, Edinburgh, September 1999.

Koksal, P., Cingil, I., Dogac, A., "A Component-Based Workflow System with Dynamic Modifications", In Proc. of Next Generation Information Technologies and Systems (NGITS '99), Springer-Verlag, Lecture Notes in Computer Science, 1649, Israel, July 1999, pp238-255.

Hagen, C., Alonso, G., "Highly available Process Support Systems implementing backup mechanisms", 18th IEEE Symp on Reliable Distributed Systems (SRDS'99), Lausanne, Switzerland, October, 1999.

Schuldt, H., Popovici, A., Schek, H.-J., "Give me all I pay for – Execution Guarantees in Electronic Commerce Payment Processes" , In Proc. of the Informatik'99 Workshop "Unternehmensweite und unternehmens-übergreifende Workflows: Konzepte, Systeme, Anwendungen", Paderborn, Germany, October 1999.

Schuldt, H., Schek, H.-J., Alonso, G., "Transactional Coordination Agents in Composite Systems", In Proc. of Intl. Database Engineering and Applications Symposium (IDEAS'99), Montreal, Canada, August 1999.

Schuldt, H., Alonso, G., Schek, H.-J., "Concurrency Control and Recovery in Transactional Process Management", In Proc. of Intl. ACM Symposium on Principles of Database Systems (PODS'99), Philadelphia, Pennsylvania, USA, June 1999.

Alonso, G., Fessler, A., Pardon, G., Schek, H.-J., "Correctness in General Configurations of Transactional Components", In Proc. of Intl. ACM Symposium on Principles of Database Systems (PODS'99), Philadelphia, Pennsylvania, USA, June 1999.

Koksal, P., Arpinar, S., Dogac, A., "History Management in Workflow Systems", International Symposium on Computer and Information Sciences, Antalya, Turkey, October 1998.

A project web site has been set up by the project leader.
http://www.srdc.metu.edu.tr/mariflow/index.html

Individual web pages have been setup by various partners (e.g.
http://research.germanlloyd.de/Projects/MARIFLOW/mariflow.html, ...)
http://www.balance-bremen.de/balance/mariflow_e.htm

The project scope has been presented to the members of the European Marine STEP Association at their regular meetings.

Distributed and Parallel Databases 7, 199–248 (1999)

Formalization of Workflows and Correctness Issues in the Presence of Concurrency*

İSMAİLCEM BUDAK ARPINAR budak@srdc.metu.edu.tr
UĞUR HALICI halici@rorqual.cc.metu.edu.tr
SENA ARPINAR nural@srdc.metu.edu.tr
ASUMAN DOĞAÇ asuman@srdc.metu.edu.tr
Software Research and Development Center, Department of Computer Engineering, Middle East Technical University (METU), 06531, Ankara, Turkey

Recommended by: Ahmed Elmagarmid

Abstract. In this paper, main components of a workflow system that are relevant to the correctness in the presence of concurrency are formalized based on set theory and graph theory. The formalization which constitutes the theoretical basis of the correctness criterion provided can be summarized as follows:

- Activities of a workflow are represented through a notation based on set theory to make it possible to formalize the conceptual grouping of activities.
- Control-flow is represented as a special graph based on this set definition, and it includes serial composition, parallel composition, conditional branching, and nesting of individual activities and conceptual activities themselves.
- Data-flow is represented as a directed acyclic graph in conformance with the control-flow graph.

The formalization of correctness of concurrently executing workflow instances is based on this framework by defining two categories of constraints on the workflow environment with which the workflow instances and their activities interact. These categories are:

- Basic constraints that specify the correct states of a workflow environment.
- Inter-activity constraints that define the semantic dependencies among activities such as an activity requiring the validity of a constraint that is set or verified by a preceding activity.

Basic constraints graph and inter-activity constraints graph which are in conformance with the control-flow and data-flow graphs are then defined to represent these constraints. These graphs are used in formalizing the intervals among activities where an inter-activity constraint should be maintained and the intervals where a basic constraint remains invalid.

A correctness criterion is defined for an interleaved execution of workflow instances using the constraints graphs. A concurrency control mechanism, namely Constraint Based Concurrency Control technique is developed based on the correctness criterion. The performance analysis shows the superiority of the proposed technique. Other possible approaches to the problem are also presented.

Keywords: workflow management system, workflow, activity, basic constraint, inter-activity constraint, time intervals, correctness, concurrency control

*This work is partially supported by the European Commission, Project No.: DC97-2496MARIFLOW, by the Middle East Technical University, the Graduate School of Natural and Applied Sciences, Project No.: AFP-97-07-02-08, and by the Scientific and Technical Research Council of Turkey, Project No.: 197E038.

1. Introduction

Today, economic imperatives are forcing enterprises to look for new information technologies to streamline their business processes. Key requirements include integrating heterogeneous information resources of an enterprise, and automating mission-critical applications that access shared information resources. Many of the activities in these enterprises are of long-duration and consist of multiple operations executed over (possibly) heterogeneous systems with very diverse response times. As a consequence of these trends, Workflow Management Systems (WFMSs) are quickly becoming the technology of choice to implement large and heterogeneous distributed execution environments where sets of interrelated activities can be carried out in an efficient and closely supervised fashion [4]. There is also a standardization effort in this respect. The Workflow Management Coalition (WfMC), an industry consortium aims at a unified terminology and a standardization of key components of a workflow management system. The WfMC identified a set of six primitives with which it is possible to describe control-flow and hence construct a workflow specification [37].

A *workflow process* is defined as a collection of processing steps (activities) organized to accomplish some business processes. An activity can be performed by one or more software systems or machines (e.g., instruments or robots), by a person or a team, or a combination of these. A workflow process contains a collection of activities and defines the order of activity invocations or condition(s) under which activities must be invoked (i.e., control-flow) and also data-flow between the activities. Activities within a workflow can themselves again be a workflow. Furthermore, an activity may be further composed of several calls to local systems (such as in multidatabases [29, 32]), and this fact is hidden at the workflow level.

The activities could be *transactional* or *non-transactional*. Transactional activities are those that access data controlled by Resource Managers (RMs) with transactional properties (i.e., ACID). These activities minimally support the atomicity property and maximally support all ACID properties of traditional transaction models [42]. These activities typically include those that interact with a DBMS by using *Commit* and *Abort* operations, stored procedures, and two-phase commit (2PC) activities.

Non-transactional activities access data controlled by RMs without transactional properties. These non-transactional processing entities include file systems, humans, legacy systems, HTTP servers, word processors, and spreadsheets. Yet, it may be possible to introduce some transactional properties to these systems, for example by wrapping non-transactional RMs to provide transaction and concurrency control services. There is further work describing how to handle non-transactional activities in [44].

1.1. Correctness issues in WFMSs

As discussed briefly in [28], the person who implements an activity is responsible for ensuring that the activity produces correct results if it is executed alone. However since workflows are long running processes, having the activities terminate (e.g., commit) within the scope of a workflow instance is an accepted practice. Thus, the data modified by these activities becomes accessible to the other activities within the same workflow instance as well as to the other workflow instances which may cause inconsistencies due to improper interleavings. Yet many scenarios in the operation of a workflow system require the preservation of consistency of at least some data items. Therefore the workflow execution must address the following two correctness concerns: (i) The correctness of concurrent executions of activities belonging to the same workflow instance; (ii) the correctness of concurrent executions of activities belonging to different workflow instances.

For example consider an *Order Processing* workflow in a manufacturing enterprise. In the processing of the *Order Processing* workflow, raw material stock is checked through a *CheckStock* activity to see whether there is enough raw material in the stock to process the order. If not, the missing raw materials are ordered from external vendors and inserted into stock through an *InsertStock* activity. Yet later in the process when the actual manufacturing is to start for this workflow instance there may not be enough raw material in the stock to process this order, because a concurrently running instance of the same or other workflows might have updated the stock. Of course, executing all these activities within the scope of a single transaction might have solved these problems but workflow systems are there to prevent the inefficiency of long-running transactions.

Another example to the data inconsistency problems is as follows: Consider the *Withdraw-Deposit* activities of a simple workflow in a bank involving two branches. *Withdraw* activity withdraws the given amount of money from an account at a branch, and the *Deposit* adds this amount to an account at another branch. Let us consider *Audit* activities of another workflow which checks the balance of these accounts. If *Withdraw-Deposit* activities and *Audit* activities are interleaved incorrectly *Audit* activities miss the money being transferred between the two accounts.

The current state of the art for workflows lacks a clear theoretical basis, correctness criteria and support for consistency of concurrent workflows to handle such problems [59]. In this paper exactly these issues are addressed. We provide a theoretical basis for the formalization of workflows, and define a correctness criterion for the consistency of concurrently executing workflows based on this formalization, and present a concurrency control technique to provide the correctness.

The main contributions of the paper are as follows:

(1) A workflow in conformance with the control-flow primitives of WfMC model is formalized based on set theory and graph theory.

We start by defining a special set whose elements may also be sets, called a nested hyperSet, and use this set in representing the conceptual groupings of activities in a workflow system. The control-flow is imposed on this set by introducing the related edges and the resulting graph is called hyperNodeGraph. Split and join nodes are introduced into this graph from where control-flow splits into multiple branches and merges into a single flow later respectively. Data-flow in a workflow is represented through a simple directed acyclic graph which is in conformance with the control-flow graph. Having thus set the necessary background, we provide a formal definition of a workflow.

(2) This formalization is used in defining a correctness criterion for concurrently executing workflows based on the semantic information available.

Workflow activities access resources which denote the set of all objects constituting the workflow environment. We define correct execution of activities in terms of their input and output conditions, which are the sets of constraints on the workflow environment. An input condition may involve two types of constraints: basic constraints that specify the correct states of a workflow environment and inter-activity constraints that define the semantic dependencies between activities, such as an activity requiring the validity of a constraint that is set or verified by a preceding activity. For example a basic constraint can state that the money being transferred between two branches of a bank through a *Withdraw-Deposit* workflow should not be destroyed during this transfer. This basic constraint remains invalid between the executions of *Withdraw* and *Deposit* activities for obvious reasons. Furthermore, consider *InsertStock* activity in the manufacturing example. Since the resulting amount of raw materials after the termination of *InsertStock* must remain in the stock until the beginning of manufacturing process that ordered it, this requirement is represented as an inter-activity constraint between *InsertStock* and the activity which is responsible from actual manufacturing process.

The intervals among activities where an inter-activity constraint should be maintained and the intervals where a basic constraint remains invalid are formalized through the graphs corresponding to these constraints. These graphs are then used in developing a correctness criterion for interleaved execution of workflows which is formally represented through a complete execution history. Simply stated, the correctness criterion requires two conditions to hold:

(i) The inter-activity constraints should be preserved in the related intervals by preventing the activities that invalidate these constraints from executing.
(ii) The activities that require the correctness of related basic constraints should be prevented from executing during the intervals where these constraints do not hold.

(3) A correctness technique, namely Constraint Based Concurrency Control (CBCC) technique, is developed based on this correctness criterion.

CBCC technique which is based on locking in conjunction with validation, controls activity interleavings in such a way that two conditions above hold. Note that this locking differs from database locking fundamentally in a way that the constraints rather than data items are locked. In this way, the disadvantages of locking data items for long duration transactions are avoided. The inter-activity constraints are locked during the time interval where they should remain valid in the shared mode. An activity that falsifies these constraints acquire a lock in the conflicting mode (i.e., exclusive mode). Through these conflicting locks activities that falsify inter-activity constraints are prevented from executing. If more than one activity require the same inter-activity constraint to be true at the overlapping time intervals, their locks do not conflict. Similarly, activities that falsify the same constraint at the overlapping time intervals do not conflict either. Note that we use the term "exclusive lock" differently than its conventional meaning in that, two exclusive locks on the same constraint do not conflict with each other in our approach.

Some activities on the other hand *may* falsify inter-activity constraints depending on the instantiation of the variables in the constraints and in their parameters. For the activities

that *may* falsify inter-activity constraints, we prefer to use an optimistic scheme rather than locking with the intention of increasing the performance, since there is a probability that the activity will not falsify these constraints. If these constraints evaluate to true at the end of an activity, the activity is allowed to terminate, otherwise it is aborted and resubmitted. Continuing with the example provided, since raw materials may be withdrawn from the stock by the concurrently executing *WithdrawFromStock* activities of some other workflows, the inter-activity constraint between *InsertStock* and the manufacturing activity may be invalidated. To prevent this, *InsertStock* obtains a shared lock on this constraint which will be released by the manufacturing activity and if a *WithdrawFromStock* activity is executed between them it goes through a validation phase.

However, it is also possible to use a more conservative approach in which activities acquire locks on the inter-activity constraints they *may* falsify in addition to the constraints they certainly falsify. We call this conservative technique based solely on locking as Constraint Locking Concurrency Control (CLCC) technique. For example, *WithdrawFromStock* activity can obtain an exclusive lock on the inter-activity constraint in CLCC technique instead of going through a validation phase.

The basic constraints specify the correct states of a workflow environment but they can be invalidated by an activity to be revalidated later through an activity or through a set of activities. The activities that require the validity of these basic constraints should not be allowed to execute in the interval where the basic constraints remain invalid, and for this purpose exclusive locks are placed on the basic constraints during these intervals by the activities that falsify these constraints. On the other hand, the activities that require the validity of the basic constraints acquire locks in the conflicting mode (shared mode). For example, *Withdraw* activity obtains an exclusive lock on the basic constraint which it falsifies, and this lock is released after *Deposit* activity terminates. Since *Audit* activity needs a shared lock on the same constraint, its execution is prevented between *Withdraw* and *Deposit* activities. The shared locks of activities which require correctness of the same basic constraint at the overlapping time intervals do not conflict with each other, and the same is true for the exclusive locks of activities which falsify the same basic constraint at the overlapping time intervals.

(4) A performance analysis of the CBCC and CLCC techniques is presented.

A performance comparison of the proposed techniques with some other approaches to the problem is also presented. The performance analysis performed through simulation indicates that our techniques result in better performance than the others.

In the work presented in this paper, semantic information about activities and workflow environment is used. In the case where this semantic information is not available, activities should be treated as black boxes and since isolation of a whole workflow execution is unacceptable because of performance reasons, smaller units of isolation should be discovered. The individual activities of a workflow are isolated by concurrency control mechanisms of local systems, and hence the main concern is to observe the concurrency control requirements between these individual activities and satisfy these requirements when required. These requirements may be determined by checking the data and control-flow dependencies between the activities. These dependencies are available at design-time, and therefore

spheres of isolations each of which includes a subset of activities of a workflow can be determined in advance and correctness of workflows can be guaranteed through the isolation of these spheres. The approaches that use this idea [7, 49, 55] are explained in Section 2. It should be noted that these approaches are much more restrictive compared to the techniques presented in this paper which make use of semantic information.

After setting the research context in the first section, the paper is organized as follows: In Section 2, the related work is given. In Section 3, we present a motivating example to explain main concepts of our approach and identify the general workflow features covered by our model. Section 4 provides formal characterization of workflows in terms of data and control-flow dependencies. Section 5 defines correctness of concurrently executing workflows and activities. In Section 6, concurrency control techniques based on this correctness definition are proposed, and the performance analysis of the techniques is given. Section 7 gives concluding remarks.

2. Related work

There are some research dealing with the correctness problem of workflows, but neither a widely accepted correctness notion nor a correctness mechanism have been reported in the literature. In the following, we confine ourselves to summarizing the related research in workflow management systems and transaction processing systems. And in spite of this research, most commercial WFMSs provide very limited capabilities for correctness and concurrency control issues [52].

In the ConTract model [51, 56], the user is given the sole responsibility for maintaining the consistency of the database with which activities interact. In order for activities to work correctly, predicates named as entry and exit invariants are defined to hold on the database. At run-time, these predicates are verified before and after an activity respectively. If entry or exit invariant evaluate to false, a conflict resolution algorithm is executed and this may involve changing values of objects in the predicates in such a way that they are satisfied. However, an inevitable result may be cancellation of activity and compensation of some previously terminated activities.

In [14] to ensure data consistency, semantic serializability of workflows is proposed as the correctness criterion. A human expert declares a compatibility matrix for activities of a workflow. Compatibility of two activities means that the ordering of two activities in an execution history is insignificant from an application point of view. If two activities are not defined as compatible they are in conflict. An execution history is semantically serializable if an equivalent serial execution exists with the same ordering of conflicting activities.

In [3], the consistency is specified using compatibility relations between sequences of activities instead of between individual activities. For instance, how different workflow instances should be interleaved is given as a matrix. The main idea is based on signatures of workflow instances that they leave on the objects they access. This signature specifies which other workflows are allowed to access the object.

In Transaction Specification and Management Environment (TSME) [27] using a transaction specification language, correctness as well as state dependencies can be specified between the activities of workflows. Different correctness dependencies such as *serializability*, *temporal*, and *cooperative dependencies* can be specified. To define conflicts,

each object is associated with a conflict table. *Serialization dependencies* are specified as acyclic serialization order dependencies between activities. *Temporal order dependencies* are specified by giving specific serialization orders between the activities. *Cooperation* between activities is provided by using breakpoints or augmenting conflict tables of shared objects.

In [7], activities are treated as black boxes and to determine concurrency control requirements between activities, data and control-flow dependencies between them are analyzed at design-time. Using this information *spheres of isolation*, each of which involves a subset of activities in a workflow, are determined and the notion of correctness is based on the isolation of these spheres. Furthermore, a technique to handle correctness of hierarchically structured workflows consisting of compound activities is proposed in [7]. In [49, 50], *M-serializability* is defined as a correctness criterion for concurrent execution of workflows. In this model, related activities of a workflow are grouped into *execution-atomic units*. *M-serializability* assumes that an activity involves a single site and it requires that activities belonging to the same *execution-atomic unit* of a workflow have compatible serialization orders at all sites they access. A similar approach is proposed in [55]. In this work, a set of activities are grouped into a *consistency unit* and traditional correctness techniques are used to provide serializable execution of this unit.

In [8], workflows are treated as multidatabase transactions and a limited form of correctness is defined. The correctness criterion requires a consistent ordering on serialization events of activities belonging to a given workflow.

In [47], a formal graph-based workflow model (ADEPT) is presented. However this work is related with preserving structural correctness of running workflow instances when their structures are modified.

2.1. *Semantics based concurrency control*

Although semantics based concurrency control mechanisms do not directly cover workflow correctness, they are related to the approach proposed in this paper. Semantics based concurrency control protocols can be broadly classified into three categories depending on whether they are based upon the semantics of transactions or upon the semantics of objects or both as described in [1]: Approaches of Garcia-Molina [25], Lynch [43], Weikum [57], Beeri [11], Farrag and Ozsu [23] can be classified into first category; works of Harder [33], Korth and Speegle [41], Herlihy [35], Badrinath and Ramamritham [10] mainly fall into second category. The works in the third category use the advantages of both approaches to increase concurrency. In [1], three semantics based correctness criteria are proposed. In [5] and [13], formal methods to decompose a transaction into smaller units using transaction and object semantics are described.

3. A motivating example

In this section, an order processing example in a highly automated manufacturing enterprise is provided using the workflow definition language of METUFlow [7, 17, 30, 38, 39] (METUFlow project has evolved to MARIFLOW project).

```
DEFINE_PROCESS OrderProcessing()
  ...
  GetOrder(OUT productNo, OUT quantity, OUT dueDate, OUT orderNo,
           OUT customerInfo)
  EnterOrderInfo(IN productNo, IN quantity, IN dueDate, IN orderNo)
  CheckBillofMaterial(IN productNo, OUT partList)
  PAR_AND (part = FOR EACH partList)
    SERIAL
      DetermineRawMaterial(IN part.No, IN part.Quantity, OUT rawMaterial,
                           OUT required)
      CheckStock(IN rawMaterial, IN required, OUT missing)
      IF (missing > 0) THEN
        VendorOrder(IN rawMaterial, IN missing)
      WithdrawFromStock(IN rawMaterial, IN required)
      GetProcessPlan(IN part.No, OUT processPlan, OUT noofSteps)
      i:=0
      WHILE (i < noofSteps)
        Assign(IN processPlan[i].cellId, IN orderNo, IN part.No,
               IN part.Quantity, IN rawMaterial, IN required)
      END_WHILE
    END_SERIAL
  END_PAR_AND
  AssembleProduct(IN productNo)
  ...
  Billing(IN orderNo, IN productNo, IN quantity, IN customerInfo)
  ...
END_PROCESS
```

Figure 1. Order processing example.

An incoming customer request causes a product order to be created and inserted into an order entry database by *GetOrder* and *EnterOrderInfo* activities respectively (figure 1). The next step is to determine required parts to assemble the ordered product by *CheckBillof-Material* activity. A part is the physical object which is fabricated in the manufacturing system. For each part, *DetermineRawMaterial* activity is executed to find out the raw materials required to manufacture that part, and a *CheckStock* activity is initiated afterwards to check stock database for the availability of these raw materials. If the required amounts of these raw materials do not exist in the stock, they should be ordered from the external vendors through *VendorOrder* (figure 2). After all missing raw materials are obtained, required raw materials to fabricate the part are withdrawn from the stock to be sent to the manufacturing cells. This is accomplished by *WithdrawFromStock* activity by decrementing the available amount of the withdrawn raw material (i.e., *quantity*(m)) in the stock database. The required steps to manufacture a part, and the manufacturing cells where these steps are performed are obtained as a result of *GetProcessPlan*. Actual manufacturing activity is initiated by assigning the work to the corresponding cells for each step in *Assign*. Finally, manufactured parts are assembled to form the product that the customer had ordered by the activity *AssembleProduct*. Further downstream activities include a billing activity. *Billing*


```
DEFINE_PROCESS VendorOrder(IN rawMaterial, IN missing)
  ...
  SendOrder(IN rawMaterial, IN missing, OUT shipmentNo)
  SuppliesArrival(IN shipmentNo)
  InsertStock(IN rawMaterial, IN missing)
END_PROCESS

DEFINE_PROCESS GetProcessPlan(IN part.No, OUT processPlan, OUT noofSteps)
  ...
  DetermineNoofCells(IN partNo, OUT cellNo)
  SelectBestCells(IN cellNo, OUT qualifiedCells)
  ConstructProcessPlan(IN qualifiedCells, OUT processPlan, OUT noofSteps)
END_PROCESS

DEFINE_PROCESS Billing(IN orderNo, IN productNo, IN quantity, IN customerInfo)
  ...
  Payment(IN orderNo, IN productNo, IN quantity, IN customerInfo, OUT amount,
          OUT paymentStatus)
  IF (paymentStatus = unpaid) THEN
    UpdateUnpaidBalance(IN customerInfo, IN amount, OUT unpaidBalance, OUT U)
    IF(unpaidBalance > U) THEN
      XOR
        RejectShipping(IN orderNo)
        MoreCredit(IN customerInfo, IN unpaidBalance, IN U)
      END_XOR
  END_IF
END_PROCESS
```

Figure 2. Order processing example (cont.)

itself is another workflow which is responsible for collecting bills of ordered products. The details of *Billing* workflow is explained in Section 5.

We further consider two other workflows defined in the system (figure 3): *Warehouse-Allocation* and *StockControl*. *WarehouseAllocation* distributes raw materials among different warehouses and reallocates the materials according to demand and delivery schedules. *RetrieveMaterial* retrieves the given amount of raw material from the stock of the source warehouse and *UpdateMaterialLocation* transfers these raw materials to the stocks of the destination warehouses in *destList*. *StockControl* workflow checks the available raw materials of different types in stocks of all warehouses through *WarehouseEvaluation* activity and prints a stock report.

4. Formal characterization of workflows

In this paper, we first attempt to formalize the correctness issues of workflow systems in the presence of concurrency and then provide a correctness technique based on the theory developed. In order to formalize the correctness issues, we first formalize the related concepts of workflows.

```
DEFINE_PROCESS WarehouseAllocation()
   ...
   GetAllocationOrder(OUT rawMaterial, OUT quantity, OUT source, OUT destList)
   RetrieveMaterial(IN rawMaterial, IN quantity, IN source)
   PAR_AND (destination = FOR EACH destList)
      UpdateMaterialLocation(IN rawMaterial, IN quantity, IN destination)
   END_PAR_AND
END_PROCESS

DEFINE_PROCESS StockControl(IN stockDBList)
   ...
   WarehouseEvaluation(IN stockDBList, OUT materialSum)
   PrintMaterialReport(IN materialSum)
END_PROCESS
```

Figure 3. WarehouseAllocation and StockControl workflows.

Currently, specification of workflows is realized through the following types of methods [45]: Script languages, net-based methods, logic-based methods, algebraic methods, and event-condition-action (ECA) rules. Most script languages and net-based methods lack a formally founded semantics. The notable exceptions are state charts [34, 58] and Petri nets [19, 26]. For a logic-based specification, temporal logic is a commonly used method [22], e.g., computational tree logic (CTL) is used to define control-flow dependencies [9]. Similarly, ECA rules are used to specify control-flow. As a final remark, many of these methods do not have either a solid formal foundation or are often not intuitive and hard to understand. Thus, a formal yet simple formalization of workflows is needed.

A workflow in the most general sense describes groupings of activities that are executed sequentially or in parallel and defines data that may be exchanged between these activities. In formalizing a workflow, we define special graphs to express this data and control-flow information. We first define a hyperSet which represents the groupings of activities in a workflow and constitutes the basis of the graph to define the control-flow. In order to introduce control-flow relations between activities, edges are introduced into a hyperSet and then a graph which is named as a hyperNodeGraph is obtained. Data-flow between the activities is represented through a simple directed acyclic graph (DAG). Since control-flow and data-flow should be in conformance with each other, consistency relation between the graphs that represent them is defined. In our model, control-flow is not permitted to contain cycles, therefore a hyperNodeGraph is refined to a hyperNodeDAG. In addition, in order to define activities from where control-flow splits into multiple branches and merges into a single flow later, split and join nodes are introduced into a hyperNodeDAG, resulting in a split-join hyperNodeDAG.

Notice that, building the required properties of workflows through graphs in a top-down fashion with starting with the most general graph and refining it to include further properties of workflows, provides a formal and clear definition of a workflow. The solid mathematical and graph theory based foundation of this formalization make it appropriate for developing a correctness theory and a favorable reference model. It should be noted that, the primitives

defined by Workflow Management Coalition (WfMC) [37] are taken into consideration in our model.

In the following, definition of a hyperSet that reflects the groupings of activities is provided. These groupings of activities are called as execution blocks or conceptual activities. When proper control-flow edges are imposed on this set, the resulting graph shows the execution structure of the workflow process.

Definition 4.1 (*HyperSet*). A *hyperSet* S is a set whose elements are simple elements or hyperelements which are simple sets or hyperSets.

Notation: The notation $S_i \in S$ is used to denote that S_i is an element of S; the notation $S(\varepsilon_i)$ is used to denote the element ε_i of S; $size(S)$ is used to denote the number of elements in S; $simple(S)$ and $hyper(S)$ are used to denote the set of simple elements of S and the set of hyperelements of S respectively. S_i, which may be a simple element or a hyperelement, is a *subelement* of a hyperSet S, denoted as $S_i \subseteqq S$, iff $S_i \in S$ or $S_i \subseteqq S_j$ for some $S_j \in S$. The notation $\varepsilon_{(i_1,i_2,\ldots,i_{k-1},i_k)}$ is used to denote a subelement which satisfies $\varepsilon_{(i_1,i_2,\ldots,i_{k-1},i_k)} \in \varepsilon_{(i_1,i_2,\ldots,i_{k-1})} \in \cdots \in \varepsilon_{(i_1,i_2)} \in \varepsilon_{i_1} \in S$. We shall drop parentheses and comas between indexes when it is clear in the notation. The *level of set* S is zero; the elements $S_i \subseteqq S$ are called *level k elements* for which the parent is *level* $k-1$ element. The set of base elements of hyperSet S, denoted as $base(S)$, is a flat set which contains all the simple subelements of S. A hyperSet S is a *flat set* if it has no hyperelement, that is any $S_i \in S$ is a simple element.

Observe that elements in a hyperSet are not disjoint. In a workflow system however, each instantiation of the same activity type should be treated as a new element at each invocation (e.g., with different set of parameter values). Furthermore, participation of the same activity instance to more than one execution block is similar to improper nesting of blocks in a procedural language. For these reasons, a *nested hyperSet* with disjoint elements is defined, and it constitutes the nodes of the hyperGraphs to be defined for representing different components of a workflow.

Definition 4.2 (*Nested hyperSet*). A hyperSet is *nested* if $base(S_i) \cap base(S_j) = \emptyset$ for any $S_i, S_j \subseteqq S$, where $S_i \not\subseteqq S_j$ or $S_j \not\subseteqq S_i$.

Example 4.1. Let $S = \{a, \{c, \{b, d\}, \{d, f\}\}, \{e, \{d, f\}, \{g, h\}\}\}$; elements of S are $\varepsilon_1 = a$, $\varepsilon_2 = \{c, \{b, d\}, \{d, f\}\}$, $\varepsilon_3 = \{e, \{d, f\}, \{g, h\}\}$; subelements of S are $\varepsilon_{21} = c$, $\varepsilon_{22} = \{b, d\}$, $\varepsilon_{23} = \{d, f\}$, $\varepsilon_{31} = e$, $\varepsilon_{32} = \{d, f\}$, $\varepsilon_{33} = \{g, h\}$, $\varepsilon_{221} = b$, $\varepsilon_{222} = d$, $\varepsilon_{231} = d$, $\varepsilon_{232} = f$, $\varepsilon_{321} = d$, $\varepsilon_{322} = f$, $\varepsilon_{331} = g$, $\varepsilon_{332} = h$ in addition to $\varepsilon_1, \varepsilon_2, \varepsilon_3$; $base(S) = \{a, b, c, d, e, f, g, h\}$; $simple(S) = \varepsilon_1$, $hyper(S) = \{\varepsilon_2, \varepsilon_3\}$; $size(S) = 3$; $size(base(S)) = 8$. Figure 4 shows this hyperSet. $S = \{a, \{c, \{b, d\}\}, \{e, f, \{g, h\}\}\}$ is a nested hyperSet which is depicted in figure 5.

Having defined a nested hyperSet which represents individual and conceptual activities, we can now define other components of a workflow. In the definition of a workflow we use four different graphs, namely a control-flow graph, a data-flow graph, and two constraints graphs. In a control-flow graph, precedence relations between individual and conceptual activities are provided, e.g., if an activity should be started after the termination of another

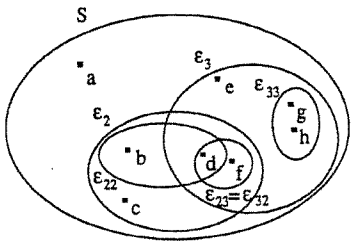


Figure 4. A hyperSet.

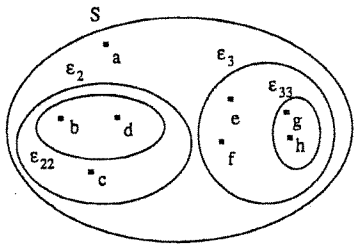


Figure 5. A nested hyperSet.

activity this is represented by a directed edge from the former activity to the latter activity in the control-flow graph. In order to represent these control-flow dependencies, we introduce edges into a nested hyperSet and thus obtain a graph which we call as a hyperNodeGraph.

Data-flow between individual activities occurs if output parameter of an activity is involved in the input parameter of a successor activity in the control-flow. Data-flow is represented through a simple directed acyclic graph (DAG) in the formalization.

In Section 5, we develop a theory in which an input condition for an activity to execute correctly is specified in terms of constraints on the workflow environment with which a workflow and its constituting activities interact. The intervals among activities where a constraint should be maintained and intervals where a constraint (may) remains invalid along a workflow execution are formalized using two constraints graphs which are 2-level hyperGraphs. Although construction and usage of a 2-level hyperGraph are explained in detail in Section 5, its definition is provided here for the sake of completeness. Furthermore, to keep the formalization at a general level we also provide the definition of a hyperGraph.

Definition 4.3 (*HyperGraph, hyperNodeGraph, 2-level hyperGraph*). A *hyperGraph* $G = (S, E)$ is a directed graph in which S is a hyperSet and edges E are defined on $S \times S \cup \{S_a \times S_a\}$ for any $S_a \subseteq S$. Notice that the graph itself can be thought as a node at an abstract level. Any $S_a \in S$ is called a *node* and $S_a \subseteq S$ is called a *subnode*. A *hyperNodeGraph* is a hyperGraph $G = (S, E)$, where S is a nested hyperSet. A *2-level hyperGraph* $G = (S, E)$ is a hyperGraph, where any $S_a \in S$ satisfies $S_a \subseteq base(S)$.

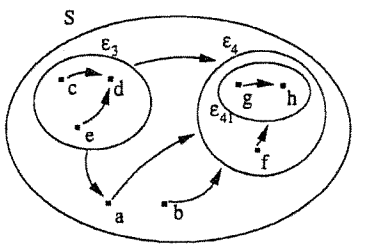


Figure 6. A hyperNodeGraph.

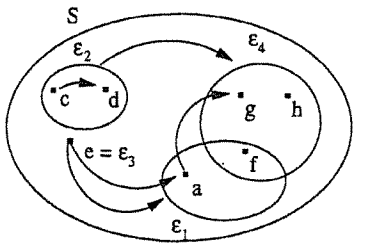


Figure 7. A 2-level hyperGraph.

In the following, these definitions are clarified through examples.

Example 4.2. Let $G = (S, E)$ be a hyperNodeGraph, where $S = \{a, b, \{c, d, e\}, \{\{g, h\}, f\}\}$ is a nested hyperSet and $E = \{\langle \varepsilon_3, \varepsilon_1\rangle, \langle \varepsilon_1, \varepsilon_4\rangle, \langle \varepsilon_3, \varepsilon_4\rangle, \langle \varepsilon_2, \varepsilon_4\rangle, \langle \varepsilon_{31}, \varepsilon_{32}\rangle, \langle \varepsilon_{33}, \varepsilon_{32}\rangle, \langle \varepsilon_{42}, \varepsilon_{41}\rangle, \langle \varepsilon_{411}, \varepsilon_{412}\rangle\}$. Figure 6 demonstrates this hyperNodeGraph.

Let $G = (S, E)$ be a 2-level hyperGraph, where $S = \{\{a, f\}, \{c, d\}, e, \{g, h, f\}\}$, and $E = \{\langle \varepsilon_2, \varepsilon_4\rangle, \langle \varepsilon_3, \varepsilon_1\rangle, \langle \varepsilon_3, \varepsilon_{11}\rangle, \langle \varepsilon_{11}, \varepsilon_{41}\rangle\}$. This graph is shown in figure 7.

Observe that the difference between a hyperGraph and hyperNodeGraph is that a nested hyperSet constitutes the nodes of a hyperNodeGraph. Therefore, only edges between the simple or hyperelements at the same level are possible. In this way, when we use a hyperNodeGraph to specify control-flow, anomalies in precedence relations are prevented. For example, if control splits into several flows and these flows are joined together within a hyperNode, control-flow can not jump into the middle of such flows from outside of this hyperNode.

Notice that level of elements in S is not greater than 2 in a 2-level hyperGraph $G = (S, E)$, i.e., level of S is 0, level of a $S_i \in S$ is 1, and level of a $S_j \in S_i$ is 2.

In a workflow, data-flow should be in conformance with its control-flow, that is, there can be data-flow between two activities only when there is a control-flow between them. Therefore a directed acyclic graph (DAG) which represents data-flow should be consistent with the hyperNodeGraph which represents corresponding control-flow. Informally, a DAG

is said to be consistent with a hyperNodeGraph iff for any edge between the two nodes of a DAG, there corresponds an edge between the same nodes or hyperNodes that include them in the transitive closure of the hyperNodeGraph. Transitive closure of a hyperNodeGraph $G = (S, E)$, denoted as $G^* = (S, E^*)$, can be obtained by taking transitive closure of the simple directed graphs obtained by abstracting the hyperNodes (abstraction of a node in a hyperNodeGraph is defined later in this section) within every hyperNode of the graph. A more formal definition can be found in [6].

Furthermore, a 2-level hyperGraph which represents constraints graphs of a workflow should be consistent with its control-flow. The reason behind this requirement is explained in Section 5.

Definition 4.4 (*Consistency with a hyperNodeGraph*). A DAG $D = (T, V)$ is said to be *consistent* with a hyperNodeGraph $G = (S, E)$ iff the following condition is satisfied:

- For any $\langle T_a, T_b \rangle \in V$, $\exists \langle S_i, S_j \rangle \in E^*$, where $S_i = T_a$ or $S_i = T_A$ such that $T_a \subseteq T_A \subseteq S$, and $S_j = T_b$ or $S_j = T_B$ such that $T_b \subseteq T_B \subseteq S$.

A 2-level hyperGraph $D = (T, V)$ is said to be *consistent* with a hyperNodeGraph $G = (S, E)$ iff for any $\langle T_k, T_l \rangle \in V$ the following condition is satisfied:

- For any $T_a \in T_k$ and $T_b \in T_l$, $\exists \langle S_i, S_j \rangle \in E^*$, where $S_i = T_a$ or $S_i = T_A$ such that $T_a \subseteq T_A \subseteq S$, and $S_j = T_b$ or $S_j = T_B$ such that $T_b \subseteq T_B \subseteq S$.

In the following, we introduce various useful operations on a nested hyperSet and a hyperNodeGraph. With these operations it becomes possible to focus on a hyperNode representing an execution block in a control-flow and conversely simplify it when its internals are not in the scope of our consideration.

A *restriction* of a hyperNodeGraph $G = (S, E)$ to one of its subelements $S_a \subseteq S$, denoted as $G(S_a)$, results in a new hyperNodeGraph which involves the node itself, its constituting simple and hyperNodes if they exist and edges between them. The other nodes and edges in the hyperNodeGraph are omitted. Figure 8 depicts the restriction of hyperNodeGraph in figure 6 to node ε_4.

Abstraction of a subelement S_a in a nested hyperSet S, denoted as S/S_a, is the replacement of S_a with an abstract simple element s_a in S. Let $S = \{a, b, \{c, d, \{e, f\}\}, \{g, h\}\}$. Abstraction of $S_3 = \{c, d, \{e, f\}\}$ in S results in $S/S_3 = \{a, b, s_3, \{g, h\}\}$, where s_3 is representing S_3.

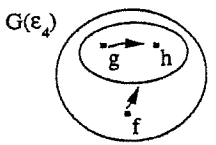


Figure 8. Restriction of the hyperNodeGraph in figure 6 to node ε_4.

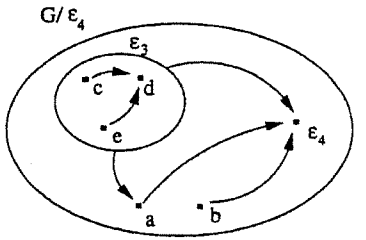


Figure 9. The abstraction of node ε_4 in hyperNodeGraph of figure 6.

An *abstraction* of a node S_a in a hyperNodeGraph G results in a new graph G/S_a, in which the node under consideration is replaced with a simple node and every edge involving the former node is replaced with a new edge involving the simple node. Figure 9 shows the abstraction of node ε_4 in hyperNodeGraph of figure 6.

Some workflow models assume that, all the structural components (i.e., control-flow) can be specified in advance. However, in some workflow applications either the number of activities in a workflow execution or the control-flow dependencies that must be enforced can not be determined in advance. These cases are named as *domain uncertainty* and *structural uncertainty* respectively [54]. Structural uncertainty occurs due to the fact that a workflow specification can contain a condition to allow selections. Our formalization covers this type of uncertainty and this is explained later in this section. Domain uncertainty occurs due to the loops (i.e., iterations) that can occur in a workflow specification. Within a loop workflow activities are repeated as long as a certain condition holds. However, representing loops in a control-flow makes the notation used in the correctness theory complicated. This is due to the fact that each instantiation of an activity within a loop should be treated as a different element for the correctness. Therefore for the sake of simplicity, we assume that a control-flow graph does not contain cycles. With this assumption a hyperNodeGraph is refined to a hyperNodeDAG in the following definition.

Definition 4.5 (*HyperNodeDAG*). A *hyperNodeDAG* is a hyperNodeGraph $G = (S, E)$ in which the abstraction of all elements results in a simple DAG, and this is recursively valid for any $S_a \subseteq S$.

Example 4.3. The hyperNodeGraph in figure 6 is a hyperNodeDAG.

Recall that, a 2-level hyperGraph representing constraints graphs of a workflow should be consistent with the control-flow graph. Since we use a hyperNodeDAG to represent the control-flow, if a 2-level hyperGraph is consistent with this graph it should be acyclic also intuitively, i.e., it should contain no cycles involving its hyperNodes or simple nodes. In this case we name this graph as a *2-level hyperDAG*. A definition of a 2-level hyperDAG is provided in [6].

In the following we provide a path definition for a hyperNodeDAG.

Definition 4.6 (A path in a hyperNodeDAG). In a hyperNodeDAG $G = (S, E)$, a *path* is a sequence $(e_1, e_2, \ldots, e_k)$ of edges such that $e_i = \langle s_i, s_{i+1} \rangle \Leftrightarrow \langle S_i, S_{i+1} \rangle \in E$, where $i = 1, \ldots, k$ and s_i, s_{i+1} are the abstractions of the nodes $S_i, S_{i+1} \in S$ respectively. A path connecting the nodes s_1 and s_{k+1} is denoted as $\langle s_1, s_{k+1} \rangle$-*path*.

A path definition makes it possible to identify a sequence of individual and conceptual activities which are executed one after another. For example, consider the conditional branches in a workflow specification. The possible flows between a split activity and a join activity can be specified as a set of paths between these activities.

In the following definition, we distinguish initial, final, first, and last nodes of a hyperNodeDAG. These nodes shall correspond to the specialized activities of a workflow. Initial and final nodes are simple nodes for which hyperNodes that include them and themselves have no predecessors and no successors respectively. Furthermore, if there is a unique initial or a unique final node they are called as first and last nodes respectively. As we provide later in this section, we require a control-flow to include unique initial and final activities, i.e., it should include a first and a last activity.

Definition 4.7 (Initial, final, first, last nodes). A simple node $\varepsilon_{in} \in S$ of a hyperNodeDAG $G = (S, E)$ is called *initial*, if $indegree(\varepsilon_{in}) = 0$, and for any S_a such that $\varepsilon_{in} \subseteq S_a$, $indegree(S_a) = 0$. A simple node $\varepsilon_{fin} \in S$ of a hyperNodeDAG $G = (S, E)$ is called *final*, if $outdegree(\varepsilon_{fin}) = 0$, and for any S_a such that $\varepsilon_{in} \subseteq S_a$, $outdegree(S_a) = 0$. If *initial* (*final*) node of a hyperNodeDAG $G = (S, E)$ is unique, it is the *first* (*last*) node of S, denoted as ε_f (ε_l).

As mentioned previously, workflow activities can be executed sequentially or in parallel. In representing control-flow, the node where the control splits into multiple parallel activities is referred to as *split node*. The node where control merges into one activity is referred to as *join node*. We introduce split and join nodes into a hyperNodeDAG definition to model these issues; the resulting graph is called a split-join hyperNodeDAG.

Definition 4.8 (Split, join nodes, split-join hyperNodeDAG). A *split node* of a hyperNodeDAG $G = (S, E)$ is a simple node $S(\varepsilon_s)$ (i.e., $\varepsilon_s \in S$) for which $indegree(\varepsilon_s) \leq 1$ and $outdegree(\varepsilon_s) > 1$. A *join node* of $G = (S, E)$ is a simple node $S(\varepsilon_j)$ (i.e., $\varepsilon_j \in S$) for which $indegree(\varepsilon_j) > 1$ and $outdegree(\varepsilon_j) \leq 1$. A *split-join hyperNodeDAG* $G = (S, E)$ is a hyperNodeDAG for which the following conditions hold:

- There exist a first and a last element.
- If there is a split element this must be the first element, and there must correspond a join element to this, and this should be the last element.
- For any restriction $G(S_a)$, where $S_a \subseteq S$ the conditions above hold.

In a control-flow graph, a split node from where control splits into two or more flows in order to execute activities in parallel is called an *and-split node*. After the termination of all activities involved in these flows, control merges into a join activity and execution continues from this activity. A split node where a decision is made upon which branch to take when encountered with multiple branches is called an *or/xor-split node*. Some of

the branches following an or-split node, and exactly one of the branches following an xor-split node are selected for execution. This selection may depend on a condition. In our model, truth value of a condition is determined by an or/xor-split node (i.e., activity) and according to this value a branch (or some branches) are selected for execution. In this case, we name this condition as a *test condition* and associate it with the branch for which it is verified. More specifically, if s is an or/xor-split node and j is the corresponding join node, each of the branches between them is represented through a path between s and j, i.e., $\langle s, j \rangle$-$path_i$, and if a test condition $\mathcal{T}$ is used to select a branch, we label the corresponding $\langle s, j \rangle$-$path_i$ with $\mathcal{T}$. If a condition is not associated with a path we assume that its label is *true*, i.e., corresponding branch is selected for execution unconditionally. Furthermore, since some of the branches are selected for parallel execution starting from an or-split node, at least one of the test conditions of these branches should be true at a time. Similarly, exactly one of the test conditions of the branches following an xor-split node should be true.

Having defined adequate tools and setting the necessary background, a formal definition of a workflow can be provided. A workflow is defined as a 5-tuple with elements representing its activities, control and data-flow and constraints graphs.

Definition 4.9 (Workflow). A *workflow* W is a tuple $W = (N, CF, DF, IC, BC)$, where

- N is a nested hyperSet whose $base(N) = T \cup S \cup J \cup \{f, l\}$ where T is the set of *individual activities*, S is the set of *split activities*, J is the set of *join activities*, and f and l are the *first* and *last activities* respectively, and they are the virtual activities indicating the start and termination of a workflow respectively.
- $CF = (N, E_{CF}, L, TC)$ is a labeled split-join hyperNodeDAG on N corresponding to the *control-flow*. The labels L is a mapping from S to $\{and, or, xor\}$ representing the types of split nodes. The labels TC is a mapping from every $\langle s, j \rangle$-*path* in CF to $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_i, \ldots, \mathcal{T}_n\}$, where $s \in S$ is an or/xor-split activity, $j \in J$ is the corresponding join activity, and $\mathcal{T}_i$ is a test condition. The following condition holds for every $\langle s, j \rangle$-*path* starting from a common or/xor-split activity s: If $L(s) = or$ then $\bigvee_{i=1}^{outdegree(s)} TC(\langle s, j \rangle\text{-}path_i) \equiv true$, and if $L(s) = xor$ then $\underline{\bigvee}_{i=1}^{outdegree(s)} TC(\langle s, j \rangle\text{-}path_i) \equiv true$, where $\underline{\bigvee}$ denotes xor operator.
- $DF = (T, E_{DF})$ is a DAG indicating the *data-flow* such that DF is consistent with CF.
- $IC = (V_{IC}, E_{IC}, L_{IC})$ is a labeled 2-level hyperDAG representing *inter-activity constraints graph*.
- $BC = (V_{BC}, E_{BC}, CL_{BC}, VL_{BC})$ is a labeled 2-level hyperDAG representing *basic constraints graph*.

In the following an example is provided to clarify the definition of workflow.

Example 4.4. Figure 10 demonstrates a sample labeled split-join hyperNodeDAG which corresponds to a control-flow. In this graph, $N = \{a, b, \{c, d, e, f, g, \{h, i, j, k, l\}, m\}, n\}$, and $T = \{b, d, e, f, g, i, j, k\}$, $S = \{c, h\}$, $J = \{l, m\}$, $f = a$, $l = n$. Furthermore, $L(c) = xor$, $L(h) = and$, and $TC(\langle c, m \rangle\text{-}path_1) = \mathcal{T}_1$, $TC(\langle c, m \rangle\text{-}path_2) = \mathcal{T}_2$, $TC(\langle c, m \rangle\text{-}path_3) = \mathcal{T}_3$.

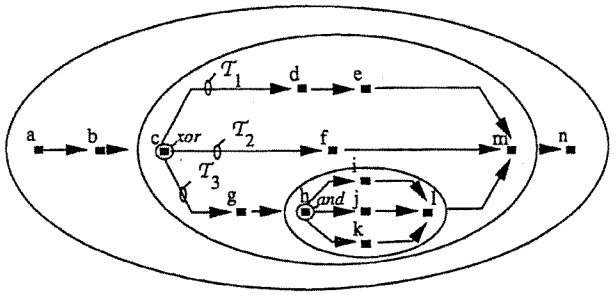


Figure 10. A labeled split-join hyperNodeDAG.

In the above workflow definition, main components of a workflow are formalized. Other properties of workflows such as assignment of agents to activities, assignment of users to roles etc. are not taken into account in the formalization, since they are out of the scope of the main focus of this work. Last two components of a workflow definition, namely inter-activity constraints graph (*IC*) and basic constraints graph (*BC*) constitute our basic building blocks to develop a correctness theory for a concurrent execution of workflows. Semantics and construction of these graphs are discussed in the following section.

5. Correctness of activities and workflows

In this section, we formalize the workflow correctness in the presence of concurrency. A workflow involves several activities each of which is performed by an agent. These activities access resources which denote the set of all objects constituting the workflow environment. We define the correct execution of activities in terms of their input and output specifications which are the set of constraints on the workflow environment. These constraints can be classified into two categories in general, namely basic constraints and inter-activity constraints which are formally defined as first-order logic formulas. The constraints that should be satisfied when an activity starts constitute the input condition of the activity. An output condition of an activity on the other hand imposes a constraint upon the workflow environment in which a workflow system must find itself after the execution of this activity.

In order to represent an interleaved execution of workflows we introduce a complete execution history and use the input and output conditions to define the correctness of this history. A complete execution history is correct if input condition of every activity involved in this history is correct when the activity starts and if the basic constraints that hold when the history starts also hold at the end of the history. We then provide a theorem which states that a complete execution history is correct if the inter-activity constraints are preserved in the required intervals and activities that require correctness of related basic constraints are prevented from executing during the intervals where these constraints do not hold. Inter-activity constraints and basic constraints are represented through inter-activity constraints graph and basic constraints graph which are used in formalizing the intervals

among activities where an inter-activity constraint should be maintained and the intervals where a basic constraint remains invalid respectively.

In the following, we provide some basic definitions and notations used in representing activity and workflow semantics and in defining the correctness of workflows. We begin by defining the state of the workflow environment.

Definition 5.1 (*Workflow environment, state of the workflow environment*). Let $RM = \bigcup_{i=1}^{n} RM^i$ be the set of transactional and non-transactional resource managers involved in a workflow system. The set of all variables (objects) controlled by RM^i is denoted by O^i. $O = \bigcup_{i=1}^{n} O^i$ denotes the set of all objects of the *workflow environment*, and $dom(o_i)$ represents the domain of an object o_i. A *state* (or *valuation*) of a workflow environment is a function $St : O \rightarrow St_\forall$, where $St_\forall = \bigtimes_{i=1}^{size(O)} dom(o_i) = dom(o_1)dom(o_2) \cdots dom(o_{size(O)})$, and $\times$ denotes the cartesian product. We use $St_\forall$ to represent the set of all possible states.

An activity t is a mapping from $St_\forall$ to $St_\forall$, i.e., $t : St_\forall \rightarrow St_\forall$. The resulting workflow environment state after an activity t is applied to state St is denoted as $t(St)$. However, this definition of an activity is not sufficient for our purposes since we require some semantic knowledge to define correctness of activities. Activity semantic is defined in terms of constraints on the workflow environment as mentioned previously.

As specification languages, first-order logic has been the dominant choice for the expression of constraints. Therefore, to represent constraints over the objects of the workflow environment we use *First-Order Logic* (*FOL*) formulas which are denoted by calligraphic letters $\mathcal{A}, \ldots, \mathcal{Z}$. More information on FOL formulas can be found in [24].

Notation: Let $\mathcal{F}$ be a FOL formula and St be a particular state of the workflow environment. We use notation $St \models \mathcal{F}$ to mean that $\mathcal{F}$ is true for the state St. If $\mathcal{F}$ is false in St this is represented as $St \not\models \mathcal{F}$. We denote the set of states that satisfy a formula $\mathcal{F}$ as $\mathcal{F}(St)$, i.e., $\mathcal{F}(St) = \{St \mid St \models \mathcal{F}\}$. The set of objects (variables) involved in a formula $\mathcal{F}$ is represented as $O(\mathcal{F})$.

Now, we can give the formal definition of a workflow activity in terms of its parameters, objects accessed, and its specification.

Definition 5.2 (*Activity*). An activity t is a tuple $t = (IP, OP, RS, WS, AS)$, where IP is the set of *input parameters*, OP is the set of *output parameters*, RS is the set of *objects read by* t, WS is the set of *objects updated by* t, AS is the *activity specification*.

In the above definition, we assume that $WS \subseteq RS$. The last item, specification of an activity, is clarified through the following definition.

Definition 5.3 (*Specification of an activity*). A *specification of an activity* t is a tuple $AS(t) = (I_t, O_t)$, where I_t and O_t are the set of FOL formulas on O (i.e., objects of the workflow environment). $\mathcal{I}_t \equiv \bigwedge_i \mathcal{I}_{t,i}$, where $\mathcal{I}_{t,i} \in I_t$, is called the *input specification* or

input condition of t and $\mathcal{O}_t \equiv \bigwedge_j \mathcal{O}_{t,j}$, where $\mathcal{O}_{t,j} \in O_t$, is called the *output specification* or *output condition* of t.

In the above definition, $\mathcal{I}_t(\mathcal{O}_t)$ is obtained by taking conjunction of all formulas in the set $I_t(O_t)$. An activity is said to be *correct* with respect to a specification $AS(t) = (I_t, O_t)$ if any terminating execution of t starting from an initial state St satisfying $\mathcal{I}_t$ ends in some final state $St' = t(St)$ satisfying $\mathcal{O}_t$, i.e., $(\forall St \in St_\forall) : ((St \models \mathcal{I}_t) \Rightarrow (t(St) \models \mathcal{O}_t))$. The activities are assumed to be correct and deterministic by intuition. More information about formal specification of programs (e.g., activities) can be found in Hoare [36], and Dijkstra's works [16]. Related work includes modal and temporal logics [22].

An output condition of an activity imposes a constraint upon the workflow environment in which workflow system must find itself after the execution of this activity. The following example demonstrates this situation.

Example 5.1. The output condition of *WithdrawFromStock* (shortly t_{WFS}) activity whose purpose is to withdraw *required* raw materials of type m_i from the stock is defined as follows:

$$\mathcal{O}_{t_{WFS}} \equiv (quantity(m_i)' = quantity(m_i) - required(m_i)). \tag{1}$$

$\mathcal{O}_{t_{WFS}}$ states that available amount of m_i is decremented by $required(m_i)$.

The input condition characterizes the set of all initial states such that the termination of an activity will leave the system in a final state satisfying the output condition. In other words, input condition of an activity represents the states of the workflow environment in which the activity can be executed correctly. Depending on the validity of the input condition, the following three possibilities can occur [16]: (1) Activation of t leads a final state satisfying $\mathcal{O}_t$; (2) activation of t leads a final state satisfying $\neg\mathcal{O}_t$; (3) activation of t does not lead a final state, i.e., activity fails to terminate properly. Since an activity t is designed correctly and it is executed in isolation, if its input condition is satisfied then the execution of t yields in first possibility. However, if the input condition is not satisfied the execution of t may result in any of three possibilities. What constitutes the input condition of an activity is described later after possible constraints in a workflow system are introduced. The following is an example to input condition of an activity.

Example 5.2. Input condition of t_{WFS} activity states that sufficient amount of raw material of type m_i should be available in the stock:

$$\mathcal{I}_{t_{WFS}} \equiv (quantity(m_i) \geq required(m_i)). \tag{2}$$

Note that, in order to satisfy the output condition in Formula 1, this input condition must be true prior to execution of t_{WFS}.

Intuitively, the following conditions should hold to execute an activity t correctly:

- t should read consistent (correct) values of objects in a workflow environment; hence, these consistent values should be displayed to the users and/or used to update other (or same) objects.
- If the correct execution of an activity depends on the validity of constraints that are set or verified by preceding activities, these constraints should still be valid prior to the execution of t.

In the following, we discuss these two conditions in detail. We start by describing what should be understood from correctness of a workflow environment. Correct states of a workflow environment are represented through basic constraints.

Definition 5.4 (Basic constraints). A *basic constraint* $\mathcal{B}_i$ is a FOL formula defined on the objects of the workflow environment. The set of all basic constraints are represented as B and called as the *basic constraints of the workflow system.* $\mathcal{B} \equiv \bigwedge_i \mathcal{B}_i$, where $\mathcal{B}_i \in B$, partition the set of all possible states $St_\forall$ into two disjoint sets, $\mathcal{B}(St)$ and $St_\forall - \mathcal{B}(St)$. First is the set of *correct states* in which all basic constraints hold, and second is the set of *incorrect states* in which one or more basic constraints are violated.

Thus, basic constraints specify the correct states of the workflow environment as the following examples demonstrate.

Example 5.3. Suppose that a basic constraint of the stock databases in the order processing example is defined as follows:

$$\mathcal{B}_1 \equiv \left(\sum_{j=1}^{w} quantity(m_{i,j}) = \mathsf{M}_i \right), \tag{3}$$

where $quantity(m_{i,j})$ represents the amount of raw material m_i in the stocks of warehouse j, and w is the total number of different warehouses in the enterprise. Total amount of raw material m_i currently residing at the stocks is denoted as M_i. Notice that, $\mathcal{B}_1$ does not prevent entering new raw materials of type m_i into stocks or withdrawing them for production; yet $\mathcal{B}_1$ implies that "raw materials should neither be created or destroyed during the transfer of these raw materials between the stocks of different warehouses by a *WarehouseAllocation* workflow".

Example 5.4. Suppose that balance of unpaid bills of a customer has a predefined upper limit. Thus, a basic constraint is defined as follows:

$$\mathcal{B}_2 \equiv ((\forall c_i \in customerList) : (unpaidBalance(c_i) \leq \mathsf{U}_i)), \tag{4}$$

where *customerList* denotes the customers of the manufacturing enterprise, and *unpaidBalance*(c_i) and U_i denote the balance of unpaid bills and the upper limit of a particular customer c_i respectively. $\mathcal{B}_2$ implies that "orders invoked by a customer should not cause an overdraft".

These examples demonstrate that basic constraints require activities to be designed and/or arranged properly in a control-flow in order to rationally update a workflow environment, so that these basic constraints are not violated during their execution. For example, activities of *Billing* workflow should be designed properly, so that balance of unpaid bills of a customer does not cause an overdraft. The restrictions induced by basic constraints in the design of a workflow are clarified later in this section through Definition 5.9.

Some activities require that some of the basic constraints must hold to execute them correctly. Thus these basic constraints are involved in the input conditions of these activities. The set of basic constraints to be involved in the input condition of an activity t is denoted as $B(t)$, and defined as follows:

$$(\forall \mathcal{B}_i \in B) : ((O(\mathcal{B}_i) \cap RS(t) \neq \emptyset) \Rightarrow (\mathcal{B}_i \in B(t))). \tag{5}$$

According to Formula 5 if an object involved in a basic constraint $\mathcal{B}_i$ is also an element of the read set of t (i.e., $RS(t)$), $\mathcal{B}_i$ is included in the input condition of t. So activity t accesses correct states of objects in the workflow environment; otherwise t may produce incorrect results or update workflow environment erroneously.

The following example demonstrates a case in which a basic constraint is included in the input condition of an activity.

Example 5.5. Consider the basic constraint $\mathcal{B}_1$ (Formula 3), and *StockControl* workflow and its *WarehouseEvaluation* (shortly t_{WE}) activity which evaluates the available raw materials of type m_i in the stocks of all warehouses. This information is printed as a report later. Since $O(\mathcal{B}_1) \cap RS(t_{WE}) = \bigcup_{j=1}^{w} quantity(m_{i,j})$, (i.e., all $quantity(m_i)$ objects in w warehouses) $\mathcal{B}_1$ should be an element of basic constraints involved in the input condition of t_{WE}, i.e., $\mathcal{B}_1 \in B(t_{WE})$. Since t_{WE} should see a correct state related to amount of raw material m_i in the stocks and $\mathcal{B}_1$ describes the corresponding set of correct states, $\mathcal{B}_1$ must hold for the correct execution of t_{WE} activity.

Assume that an incorrect state is also acceptable for a particular *WarehouseEvaluation* activity. Hence a report about approximate quantity of a raw material in the stocks is allowed. In this case, basic constraint $\mathcal{B}_1$ can be excluded from $B(t_{WE})$ although implied by the Formula 5. In this way, flexibility in the specification of incorrect but acceptable states for an activity t can be achieved. This approach resembles the *isolation levels* provided by some database management systems [31].

Although activities are usually execution-atomic (i.e., isolated) steps by their nature, there may be semantic dependencies between them that must be observed and preserved. For example, an activity may cause that a constraint to be satisfied on the workflow environment after its termination, and a successor activity may be executed with the assumption of the validity of this constraint. Furthermore, another activity may evaluate a constraint and determine its truth value, and this value may be used in the workflow specification to allow branching. Activities relying on the selected branch are likely to require validity of the constraint associated with their branch when they are executing. Both cases impose dependencies between activities. We represent such dependencies between individual activities as a set of inter-activity constraints on the workflow environment.

Definition 5.5 (Inter-activity constraints). Let $W = (N, CF, DF, IC, BC)$ be a workflow, and t_i and t_j be the particular activities of this workflow, i.e., $t_i \in base(N)$, $t_j \in base(N)$. The *inter-activity constraints* between t_i and t_j, denoted as $C_{\{t_i,t_j\}}$, is a set of constraints on the workflow environment which satisfy the following conditions:

(1) t_i precedes t_j in CF.
(2) $(\forall \mathcal{D} \in C_{\{t_i,t_j\}}) : (\mathcal{D} \in I_{t_j})$.
(3) $(\forall \mathcal{D} \in C_{\{t_i,t_j\}}, \exists \mathcal{F} \in O_{t_i}) : (\mathcal{F} \Rightarrow \mathcal{D})$.

In the above definition, if a constraint $\mathcal{F}$ in the output condition of a preceding activity t_i implies a constraint $\mathcal{D}$ in the input condition of a successor activity t_j, the latter constraint is included in the set of inter-activity constraints between these two activities. Note that we require implication instead of equivalence between constraints $\mathcal{F}$, and $\mathcal{D}$. This is due to the fact that, validity of $\mathcal{F}$ already guarantees the validity of $\mathcal{D}$, and $\mathcal{D}$ is the constraint that is involved in the input condition of the successor activity. Thus the inclusion of the less restrictive constraint $\mathcal{D}$ in the set of inter-activity constraints is enough.

Notation: If the conditions in Definition 5.5 hold we say that constraint $\mathcal{D}$ is *emanating from* activity t_i and *incoming to* activity t_j. We use these terms to provide the reader the ability to pictorially imagine the constraint relations between activities. The set of inter-activity constraints incoming to and emanating from an activity t_j are denoted as $C_{in}(t_j)$ and $C_{out}(t_j)$ respectively and defined as follows: $C_{in}(t_j) = \bigcup_i C_{\{t_i,t_j\}}$, $C_{out}(t_j) = \bigcup_k C_{\{t_j,t_k\}}$. We denote the set of all inter-activity constraints in a workflow as C, i.e., $C = \bigcup_j C_{in}(t_j) = \bigcup_j C_{out}(t_j)$.

The following examples present some inter-activity constraints in the order processing example.

Example 5.6. Consider *CheckStock* (shortly t_{CS}) and *WithdrawFromStock* (t_{WFS}) activities. t_{CS} checks whether the required amount of raw material of type m_i (i.e., $required(m_i)$) to manufacture a particular part is available in the stock. Thus the current value of $quantity(m_i)$ (e.g., n) is determined and using this value the missing raw materials (i.e., $missing(m_i)$) that should be ordered from external vendors are calculated. Ordered raw materials are inserted into stock through *InsertStock* (t_{IS}) activity of *VendorOrder* workflow. Thus the output condition of t_{CS}, and input and output conditions of t_{IS} are defined as follows:

$$\mathcal{O}_{t_{CS}} \equiv ((quantity(m_i) = n) \wedge (missing(m_i) = required(m_i) - n)), \tag{6}$$

$$\mathcal{I}_{t_{IS}} \equiv (quantity(m_i) \geq n), \tag{7}$$

$$\mathcal{O}_{t_{IS}} \equiv ((quantity(m_i)' = quantity(m_i) + missing(m_i)) \wedge (quantity(m_i)' \geq required(m_i))), \tag{8}$$

where $quantity(m_i)'$ is the new quantity of m_i when t_{IS} is completed. Since output condition of t_{CS} implies input condition of t_{IS}, i.e., $\mathcal{O}_{t_{CS}} \Rightarrow \mathcal{I}_{t_{IS}}$, and output condition of t_{IS} implies input condition of t_{WFS} (Formula 2), i.e., $\mathcal{O}_{t_{IS}} \Rightarrow \mathcal{I}_{t_{WFS}}$, the constraints $(quantity(m_i) \geq n)$,

and ($quantity(m_i) \geq required(m_i)$) are included in the sets $C_{\{t_{CS},t_{IS}\}}$, and $C_{\{t_{IS},t_{WFS}\}}$ respectively. In other words, if n particular materials of type m_i are available in t_{CS}, at least this amount of material should be available in the corresponding t_{IS} also, so $quantity(m_i)$ becomes larger than or equal to $required(m_i)$ after the insertion of missing materials into stock. $Required(m_i)$ materials should remain in the stock, so $\mathcal{I}_{t_{WFS}}$ holds when t_{WFS} is executed. Notice that ($quantity(m_i) \geq n$) is an element of $C_{in}(t_{IS})$, and $C_{out}(t_{CS})$, and ($quantity(m_i) \geq required(m_i)$) is an element of $C_{in}(t_{WFS})$, and $C_{out}(t_{IS})$. Furthermore, both of these constraints are elements of C.

The following is also an example from order processing workflow to further clarify inter-activity constraints.

Example 5.7. Consider *GetProcessPlan* workflow, and its *SelectBestCells* (t_{SBC}) activity. t_{SBC} evaluates the manufacturing cells in the factory and selects the required number of the best qualified cells to manufacture a particular part. Thus,

$$\mathcal{O}_{t_{SBC}} \equiv ((\forall cell_i \in qualifiedCells, \forall cell_j \in (cells - qualifiedCells)) : (rank(cell_i) \geq rank(cell_j))), \quad (9)$$

where *qualifiedCells*, and $rank(cell_i)$ denote the set of selected cells, and rank of a particular cell respectively. The *rank* is obtained by evaluating qualifications, workload, capacity, etc. of a particular cell. *Cells* denotes the set of all operational cells in the factory. Since the selected best cells should remain so until the work is actually assigned to them in the corresponding *Assign* (t_A) activities, the input condition of a t_A activity for $cell_i$ should be defined as follows:

$$\mathcal{I}_{t_{A(cell_i)}} \equiv ((\forall cell_j \in (cells - qualifiedCells)) : (rank(cell_i) \geq rank(cell_j))). \quad (10)$$

Since $\mathcal{O}_{t_{SBC}} \Rightarrow \mathcal{I}_{t_{A(cell_i)}}$, Formula 10 should be an element of $C_{\{t_{SBC},t_{A(cell_i)}\}}$.

In order to represent inter-activity constraints graphically in a workflow, we use a special graph, namely *inter-activity constraints graph* which is a labeled 2-level hyperDAG defined in Section 4. In this way, inter-activity constraints can be represented in the way control and data-flow are represented.

Let $W = (N, CF, DF, IC, BC)$ be a workflow; inter-activity constraints between the activities of W are represented as a labeled 2-level hyperDAG $IC = (V_{IC}, E_{IC}, L_{IC})$, where V_{IC} and E_{IC} denote the nodes and edges respectively. V_{IC} is a hyperSet, and for any $S_a \in V_{IC}$, $S_a \subseteq base(N)$, and for any $\langle S_a, S_b \rangle \in E_{IC}$, $S_a \in base(N)$ and $S_b \subseteq base(N)$. L_{IC} are the labels of the edges and it is a mapping from the edges in E_{IC} to the inter-activity constraints in C. For a given set of inter-activity constraints between activity pairs, if there is a constraint $\mathcal{F}$ between t_i and t_j, this is represented through an edge $\langle t_i, t_j, \mathcal{F} \rangle$ in IC. If a constraint $\mathcal{F}$ emanating from an activity t_i is incoming to more than one activity, these activities are grouped into a hyperSet $S_{(t_i,\mathcal{F})}$ and this situation is represented through the edge $\langle t_i, S_{(t_i,\mathcal{F})}, \mathcal{F} \rangle$. The following example demonstrates the construction of an inter-activity constraints graph.

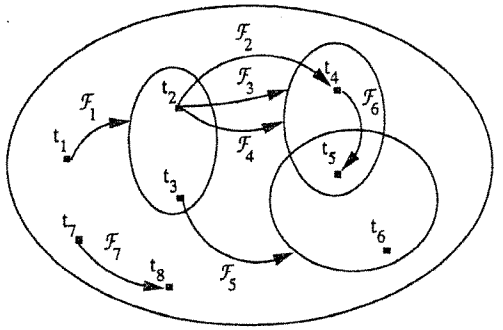


Figure 11. Inter-activity constraints graph.

Example 5.8. Let $C = \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4, \mathcal{F}_5, \mathcal{F}_6, \mathcal{F}_7\}$, and $C_{\{t_1,t_2\}} = \{\mathcal{F}_1\}$, $C_{\{t_1,t_3\}} = \{\mathcal{F}_1\}$, $C_{\{t_2,t_4\}} = \{\mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4\}$, $C_{\{t_2,t_5\}} = \{\mathcal{F}_3, \mathcal{F}_4\}$, $C_{\{t_3,t_5\}} = \{\mathcal{F}_5\}$, $C_{\{t_3,t_6\}} = \{\mathcal{F}_5\}$, $C_{\{t_4,t_5\}} = \{\mathcal{F}_6\}$, $C_{\{t_7,t_8\}} = \{\mathcal{F}_7\}$. Therefore, as explained above, t_2 and t_3 are grouped into a hyperSet and $\langle t_1, \{t_2, t_3\}, \mathcal{F}_1 \rangle$ is included in IC. Eventually, IC corresponding to C is obtained as depicted in figure 11.

Note that IC is consistent with control-flow graph (CF) due to Condition 1 of Definition 5.5.

An inter-activity constraints graph can be simplified by removing redundant edges from it. In general if an edge covers another edge in an inter-activity constraints graph and constraint of the former edge implies the constraint of the latter edge, the latter edge can be removed from the graph. This is due to the fact that if first inter-activity constraint is valid between the executions of activities in its source and sink, validity of second constraint is automatically guaranteed. Furthermore, some inter-activity constraints can be removed from an inter-activity constraints graph through human intervention. If invalidity of an inter-activity constraint is acceptable for a particular activity, the edge corresponding to this constraint can be excluded from the graph by a workflow designer. This is similar to exclusion of some basic constraints from the input condition of an activity. Details of the simplification process and elimination of constraints are provided in [6].

We use an inter-activity constraints graph to develop a correctness criterion for workflows. Since inter-activity constraints contribute to the input condition of an activity, constraints in an IC graph should be preserved between the nodes of the graph during execution of the workflow since only activities are isolated not the whole workflow.

Up to this point, we have defined *basic constraints* and *inter-activity constraints*. Having defined these two types of constraints, we can now formally provide the semantic of an input condition of an activity t as follows:

$$\mathcal{I}_t \equiv \left(\bigwedge_i \mathcal{B}_i\right) \wedge \left(\bigwedge_j \mathcal{F}_j\right) \wedge \left(\bigwedge_k \mathcal{G}_k\right), \quad (11)$$

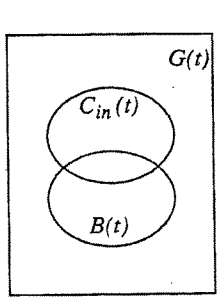


Figure 12. Relations between inter-activity, basic, and extensional constraints.

where $\mathcal{B}_i \in B(t)$, and $\mathcal{F}_j \in C_{in}(t)$, and $\mathcal{G}_k \in G(t)$. Intuitively, input condition of an activity is the conjunction of the basic constraints, inter-activity constraints, and constraints in $G(t)$ which are required to execute this activity correctly. $G(t)$ is composed of a set of constraints to execute t correctly which are not included in neither in $B(t)$ nor in $C_{in}(t)$ as depicted in figure 12. Therefore, constraints in $G(t)$ refer to state information which is not transferred from preceding activities or can not be represented through basic constraints. For example, consider *WithdrawFromStock* activity (shortly t_{WFS}) and its input condition which is defined in Formula 2. Furthermore, suppose that a *CheckStock* activity is not placed before it in the control-flow; therefore quantity of missing materials can not be determined and inserted into stock before the execution of t_{WFS}. In this case, $(quantity(m_i) \geq required(m_i))$ is not in $B(t_{WFS})$ and $C_{in}(t_{WFS})$. This type of constraints are called as an *extensional constraints*, and included in the set $G(t)$ as depicted in figure 12.

Later in this section we discuss the cases in which the constraints in the input condition of an activity are violated and therefore its correct execution is sacrificed. To detect these violations we are interested in whether an activity maintains a constraint. The following definition is provided to formalize this issue.

Definition 5.6 (*Preserve function*). Let t be an activity and $\mathcal{F}$ be a FOL formula on the workflow environment. *Preserve* $(t, \mathcal{F})$ is a three-valued function which is defined as follows:

(1) $Preserve(t, \mathcal{F}) = true$ (1) if $(\forall St \in St_\forall) : ((St \models \mathcal{F}) \Rightarrow (t(St) \models \mathcal{F}))$. In this case we say that "t preserves $\mathcal{F}$".
(2) $Preserve(t, \mathcal{F}) = false$ (0) if $(\forall St \in St_\forall) : ((St \models \mathcal{F}) \Rightarrow (t(St) \not\models \mathcal{F}))$. In this case we say that "t falsifies (or invalidates) $\mathcal{F}$".
(3) $Preserve(t, \mathcal{F}) = may\ be$ (1/2) if $(\exists St \in St_\forall) : ((St \models \mathcal{F}) \Rightarrow (t(St) \not\models \mathcal{F}))$. In this case we say that "t may falsify (or may invalidate) $\mathcal{F}$".

Intuitively, $Preserve(t, \mathcal{F}) = 0$ or $1/2$ requires that $WS(t) \cap O(\mathcal{F}) \neq \emptyset$. Result of *Preserve* $(t, \mathcal{F})$ is not always binary since the effects of an activity on the state of the workflow environment may depend on the actual values of its input parameters and/or the current values of variables in $O(\mathcal{F})$. Thus an activity may not falsify some of the constraints depending on the actual instantiation of these parameters and variables. The following is a simple example to demonstrate this situation.

Example 5.9. Let $\mathcal{F}_1 \equiv (x_1 < x_2)$, and $\mathcal{F}_2 \equiv (x_1 = x_2)$, and $t_1 = increment(x_2)$, $t_2 = decrement(x_1)$, $t_3 = increment(x_1)$, $t_4 = decrement(x_2)$. Assume that $dom(x_1)$, and $dom(x_2)$ are equal to the same totally ordered set with respect to a relation $<$. $Preserve(t, \mathcal{F}_1) = 1$ for $t \in \{t_1, t_2\}$; $Preserve(t, \mathcal{F}_1) = 1/2$ for $t \in \{t_3, t_4\}$; $Preserve(t, \mathcal{F}_2) = 0$ for $t \in \{t_1, t_2, t_3, t_4\}$.

According to the approach described above, we would like to check activities to see whether they always preserve a constraint $\mathcal{F}$. But, the recent results in the related literature show that it is almost impossible to automatically determine the value of *Preserve* for a given activity and a constraint. As noted in [12], for transactions specified as *select-project-join expressions* of relational algebra and constraints specified as FOL formulas, it is *undecidable* to check if a given transaction preserves a given constraint. Therefore, we simply assume that a workflow system administrator and/or workflow designers can specify the value of $Preserve(t, \mathcal{F})$.

As discussed previously, basic constraints specify the correct states of the workflow environment. Invalidation of basic constraints may be permissible by the individual activities; yet this situation imposes some restrictions (1) on the execution of the workflow in which an activity that invalidates (or may invalidate) a basic constraint resides, and (2) on the execution of activities which require accessing correct states. Since basic constraints represent these correct states, if they are violated during a workflow execution they should be resatisfied again prior to the termination of this execution. Otherwise the workflow environment is left in an incorrect state. Therefore, a workflow should be designed properly so that, if it includes an activity which falsifies (or may falsify) a basic constraint then it should include another activity (or possibly a set of activities) which certainly guarantees revalidation of this basic constraint. Furthermore, if the same basic constraint is involved in the input condition of another activity, execution of this activity should be prevented between the executions of former and latter activity (or activities). To capture these issues we have defined a validating set of activities for a basic constraint.

Definition 5.7 (*And, or-validating sets*). Let $W = (N, CF, DF, IC, BC)$ be a workflow, and B be the set of basic constraints of the workflow system. Furthermore, let $t_i \in T$, where $VS \subset T$, and T represents the individual activities in N. VS is an *and-validating set for* $\mathcal{B} \in B$ if the following conditions hold:

(1) $Preserve(t_i, \mathcal{B}) = 0$ or $1/2$.
(2) $(\forall t_j \in VS) : (t_i$ precedes t_j in $CF)$.
(3) $\bigwedge_j \mathcal{O}_{t_j} \Rightarrow \mathcal{B}$, where $t_j \in VS$.
(4) $(\forall t_j \in VS) : (\bigwedge_k \mathcal{O}_{t_k} \not\Rightarrow \mathcal{B})$, where $t_k \in (VS - t_j)$.

VS is an *or-validating set for* $\mathcal{B} \in B$ if the following conditions hold:

(1) Conditions 1, and 2 above.
(2) $(\forall t_j \in VS) : (\mathcal{O}_{t_j} \Rightarrow \mathcal{B})$.

Informally, VS is an and-validating set for $\mathcal{B}$ if $\mathcal{B}$ is a basic constraint which is (or may be) invalidated by t_i, and validated collectively by the elements of VS. Condition 4 guarantees that execution of activities in a subset of an and-validating set VS is not a sufficient condition for the validation of $\mathcal{B}$, and therefore VS is the minimum set of activities to validate $\mathcal{B}$. If the execution of at least one element of a set of activities (VS) is a sufficient condition for the validation of $\mathcal{B}$ we call VS as the or-validating set for $\mathcal{B}$.

Notation: We denote the set of basic constraints which are (or may be) invalid between t_i and activities of an and-validating set VS as $SB_{\{t_i, VS, and\}}$. The set of basic constraints which are (or may be) invalid between t_i and at least one activity of an or-validating set VS is denoted as $SB_{\{t_i, VS, or\}}$.

In the following, we clarify these definitions through examples.

Example 5.10. Consider the *WarehouseAllocation* workflow in figure 3. Output conditions of *RetrieveMaterial* (t_{RM}), and *UpdateMaterialLocation* (t_{UML}) activities of a *Warehouse-Allocation* workflow are defined as follows:

$$\mathcal{O}_{t_{RM(w_j)}} \equiv (quantity(m_{i,j})' = quantity(m_{i,j}) - n), \tag{12}$$

$$\mathcal{O}_{t_{UML(w_k)}} \equiv (quantity(m_{i,k})' = quantity(m_{i,k}) + l_k), \tag{13}$$

where w_j represents the source warehouse, and w_k represents a warehouse k in *destList*, i.e., $w_k \in destList$, and $\sum_{k=1}^{size(destList)} l_k = n$. Since after n raw materials of type m_i are withdrawn from the stock of warehouse j, $\mathcal{B}_1$ (Formula 3) is no longer true of the workflow environment state. However, $\mathcal{B}_1$ is resatisfied after the termination of the corresponding t_{UML} activities which distribute withdrawn amount to stocks at different warehouses in *destList*. In this case $t_{UML(w_k)}$ activities for each warehouse k constitute an and-validating set for $\mathcal{B}_1$, since after the termination of all activities in this set $\mathcal{B}_1$ is satisfied again, and therefore $SB_{\{t_{RM(w_j)}, \cup_{k=1}^{size(destList)} t_{UML(w_k)}, and\}} = \{\mathcal{B}_1\}$.

The following is an example to an or-validating set for a basic constraint.

Example 5.11. Consider *Billing* workflow and its *UpdateUnpaidBalance* (t_{UUB}), *Reject-Shipping* (t_{RS}), and *MoreCredit* (t_{MC}) activities (figure 2). Their output conditions are defined as follows:

$$\mathcal{O}_{t_{UUB}} \equiv (unpaidBalance(c_i)' = unpaidBalance(c_i) + b) \tag{14}$$

$$\mathcal{O}_{t_{RS}} \equiv ((unpaidBalance(c_i)' = unpaidBalance(c_i) - b) \wedge (orderStatus = rejected)) \tag{15}$$

$$\mathcal{O}_{t_{MC}} \equiv ((\mathsf{U}_i' = \mathsf{U}_i + c) \wedge (\mathsf{U}_i' \geq unpaidBalance(c_i))), \tag{16}$$

where U_i' denotes the new upper limit after t_{MC} is terminated. If a customer c_i does not pay the bill of an ordered product, her/his balance of unpaid bills (i.e., $unpaidBalance(c_i)$) is updated in t_{UUB} activity (Formula 14 above). Since $Preserve(t_{UUB}, \mathcal{B}_2) = 1/2$, basic constraint $\mathcal{B}_2$ (Formula 4) may be invalid at this moment. In this case either shipping of ordered product is rejected (or delayed) and $unpaidBalance(c_i)$ is decremented in t_{RS} activity (Formula 15), or if responsible branch of the enterprise grants more credit to this customer, her/his upper limit (U_i) is incremented in t_{MC} activity, thus $\mathsf{U}_i \geq unpaidBalance(c_i)$ holds (Formula 16). Observe that $\mathcal{B}_2$ is certainly satisfied after the termination of either t_{RS} or t_{MC} activity. Therefore t_{RS} and t_{MC} activities constitute an or-validating set for $\mathcal{B}_2$, and $SB_{\{t_{UUB}, \{t_{RS}, t_{MC}\}, or\}} = \{\mathcal{B}_2\}$.

As the previous examples demonstrate activities of an and/or-validating set guarantee revalidation of a basic constraint. Yet to achieve this, there is a prerequisite which is a natural outcome of our definition of activity semantic: Input conditions of activities of an and/or-validating set should hold when they are executed. Only in this way Condition 3 for an and-validating set, and Condition 2 for an or-validating set in Definition 5.7 can be satisfied. To achieve this, required inter-activity constraints between the activity which (may) invalidate a basic constraint and activities in the corresponding validating set should be preserved. The following example demonstrates this requirement.

Example 5.12. In the manufacturing example, a product is composed of parts and parts are further composed of raw materials. Therefore consistency of technical data, i.e., design information belonging to a product and its constituting parts is an essential requirement in a manufacturing process. To state this, a basic constraint of the system is defined as follows:

$$\mathcal{B}_3 \equiv ((\forall prod_i \in products, \forall part_j \in parts) : ((part_j \in P(prod_i)) \Rightarrow Consistent(design(prod_i), design(part_j)))). \tag{17}$$

According to $\mathcal{B}_3$, design of a product, i.e., $design(prod_i)$, should be consistent with designs of its constituting parts, i.e., $design(part_j)$, where $part_j \in P(prod_i)$. Let *UpdatePart-Design* (shortly t_{UPartD}) and *UpdateProductDesign* (t_{UProdD}) be two activities whose output conditions are defined as follows:

$$\mathcal{O}_{t_{UPartD}} \equiv ((design(part_j)' = design(part_j) + \Delta) \wedge Consistent(design(prod_i) + F(\Delta), design(part_j)')) \tag{18}$$

$$\mathcal{O}_{t_{UProdD}} \equiv ((design(prod_i)' = design(prod_i) + F(\Delta)) \wedge Consistent(design(prod_i)', design(part_j))) \tag{19}$$

where $design(part_j)'$ and $design(prod_i)'$ represent new designs. t_{UPartD} changes design of a part by Δ, and t_{UProdD} updates corresponding product through a function $F(\Delta)$, so that the consistency of designs for product and its part is achieved again after t_{UProdD}, i.e., $\mathcal{O}_{t_{UProdU}} \Rightarrow \mathcal{B}_3$. In order to get the above result, however, input condition of t_{UProdD} should

include the constraint $Consistent(design(prod_i) + F(\Delta), design(part_j))$. That is, prior to execution of t_{UProdD}, change made in $design(part_j)$ must remain the same (i.e., no other activities change the design of the part), so update of $design(prod_i)$ by $F(\Delta)$ should make the design of product consistent with its part again. Note that, the output condition of t_{UPartD} also includes this constraint since this part is redesigned with the assumption that the product design will change accordingly. As a result, the constraint $Consistent(design(prod_i) + F(\Delta), design(part_j))$ is included in the set of inter-activity constraints between t_{UPartD}, and t_{UProdD}, i.e., it is an element of $C_{\{t_{UPartD}, t_{UProdD}\}}$.

In Definition 5.7, it is assumed that if a basic constraint is (or may be) invalidated by a previously executed activity, its revalidation is guaranteed by successor activities in control-flow. However, this invalidation can be prevented through the execution of a preceding activity or a set of activities. More precisely, if $Preserve(t, \mathcal{B}) = 1/2$ invalidation of $\mathcal{B}$ by the execution of t can be prevented by the execution of some preceding activities in control-flow, thus $\mathcal{O}_t \Rightarrow \mathcal{B}$ [6]. The details are omitted here due to space limitations.

The presented examples provide sufficient guidance for workflow designers, so if their workflow specification includes an activity which (may) invalidates a basic constraint they should also include other activities conforming to the definitions of validating sets or prevent this invalidation by placing preceding activities.

We formally represent and/or-validating sets and intervals at which the basic constraints are (or may be) invalid during the execution of a workflow W, through a labeled 2-level hyperDAG $BC = (V_{BC}, E_{BC}, CL_{BC}, VL_{BC})$, where V_{BC}, and E_{BC} represent nodes, and edges respectively. V_{BC} is a hyperSet, and for any $S_a \in V_{BC}$, $S_a \subseteq T$ and for any $\langle S_a, S_b \rangle \in E_{BC}$, $S_a \in T$ and $S_b \subseteq T$. Recall that T is the set of individual activities of W. CL_{BC} and VL_C are the labels of edges in E_{BC}; CL_{BC} is a mapping from E_{BC} to negated elements of B, where B is the set of basic constraints of the workflow system, and VL_{BC} is a mapping from E_{BC} to $\{and, or\}$ denoting the types of validating sets. E_{BC} is constructed through the use of following principles:

- $(\forall \mathcal{B} \in B) : ((\mathcal{B} \in SB_{\{t_i, VS, and\}}) \Rightarrow (\langle t_i, VS, \neg\mathcal{B}, and\rangle \in E_{BC}))$.
- $(\forall \mathcal{B} \in B) : ((\mathcal{B} \in SB_{\{t_i, VS, or\}}) \Rightarrow (\langle t_i, VS, \neg\mathcal{B}, or\rangle \in E_{BC}))$.

According to these principles, if VS is an and-validating set or an or-validating set for $\mathcal{B}$ this situation is represented by the edges $\langle t_i, VS, \neg\mathcal{B}, and\rangle$ and $\langle t_i, VS, \neg\mathcal{B}, or\rangle$ respectively. Note that if VS includes more than one activity it is represented as a hyperSet in BC. If VS has one element, this element is represented with a simple node, and since type of VS (i.e., and/or) is immaterial in this case, label of the edge incoming to VS representing its type is omitted. Furthermore BC is consistent with control-flow graph (CF) due to Condition 2 of Definition 5.7. The following example demonstrates the construction of a basic constraints graph using the principles above.

Example 5.13. Let $B = \{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_6, \mathcal{B}_7\}$, and $SB_{\{t_1, \{t_2, t_3\}, and\}} = \{\mathcal{B}_1\}$, $SB_{\{t_2, \{t_4, t_5\}, or\}} = \{\mathcal{B}_2, \mathcal{B}_3\}$, $SB_{\{t_3, t_6\}} = \{\mathcal{B}_4, \mathcal{B}_5\}$, $SB_{\{t_5, t_6\}} = \{\mathcal{B}_5\}$. The corresponding basic constraints graph BC is depicted in figure 13.

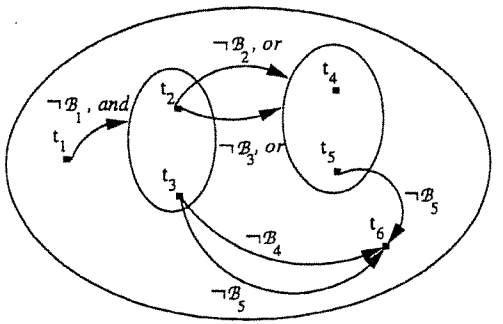


Figure 13. Basic constraints graph.

We use basic constraints graph in conjunction with inter-activity constraints graph to develop the notion of correct execution of workflows. To define a correctness criterion we need the definition of a complete execution history of workflow instances. In the following, the definition of a complete execution of a workflow is provided which is then used in defining the history.

In Section 4, control-flow of a workflow is formalized as a labeled split-join hyperNodeDAG. In this graph, or/xor-split nodes cause some activities of the workflow not to take place in the actual execution. This is due to the fact that after the execution of an or/xor-split node a decision is made upon which branch to take. To define the parts of a workflow which are actually executed at run-time, namely a complete execution of a workflow, the following algorithm is provided. In this algorithm, $G = (T_G, E_G, L_G, TC_G)$ is a labeled split-join hyperNodeDAG which is local to the algorithm itself. The split-join hyperNodeDAG $CE = (N_{CE}, E_{CE})$ is the resulting complete execution graph for a given control-flow graph, $CF = (N, E_{CF}, L, TC)$.

Algorithm 5.1 (Complete execution generation algorithm).

procedure PathGenerate(G):
begin
1. $f \leftarrow first(G/\varepsilon_1/\varepsilon_2 \ldots /\varepsilon_{size(T_G)})$, *where* $G = (T_G, E_G, L_G, TC_G)$;
2. $l \leftarrow last(G/\varepsilon_1/\varepsilon_2 \ldots /\varepsilon_{size(T_G)})$;
3. **if** f *is a split node* **then**
4. **case** $L_G(f)$ **of**
begin
5. *and* : **for** *every* $\langle f, l\rangle$-*path* $\subseteq E_G$ **do** $E_{CE} \leftarrow E_{CE} \cup \langle f, l\rangle$-*path*;
6. *or* : **for** *some* $\langle f, l\rangle$-*path* $\subseteq E_G$ **do** $E_{CE} \leftarrow E_{CE} \cup \langle f, l\rangle$-*path*;
7. *xor* : **for** *exactly one* $\langle f, l\rangle$-*path* $\subseteq E_G$ **do** $E_{CE} \leftarrow E_{CE} \cup \langle f, l\rangle$-*path*;
end
8. **else** $E_{CE} \leftarrow E_{CE} \cup \langle f, l\rangle$-*path*
end

program main:
begin
1. $N_{CE} \leftarrow \emptyset$, $E_{CE} \leftarrow \emptyset$;
2. *PathGenerate*(CF);
3. **for** *every node* $\varepsilon_{CE} \in N_{CE}$ *and* $\varepsilon_{CE} \in hyper(N)$ **do**
4. *PathGenerate*($CF(\varepsilon_{CE})$)

end

The procedure *PathGenerate* accepts a labeled split-join hyperNodeDAG G as an input. In *Steps* 1 and 2, each hyperNode of G is replaced with an abstract simple element; thus it results in a simple DAG. First and last elements of the DAG are assigned to f and l respectively. If f is an *and-split node* all paths connecting it to l are included in CE; if f is an *or-split node* some of the paths connecting it to l are included in CE; if f is an *xor-split node* exactly one of the paths connecting it to l is included in CE. If f is not a split node, single $\langle f, l\rangle$-*path* is included in CE.

The main program which calls procedure *PathGenerate* is also provided above. After initialization, this main program executes *PathGenerate* for control-flow, CF. For every node ε_{CE} included in CE after this step (i.e., $\varepsilon_{CE} \in N_{CE}$), if this node corresponds to a hyperNode in CF, *PathGenerate* is called with the restriction of CF to this node (i.e., $CF(\varepsilon_{CE})$) as the input. The program executes until there is no element in CE corresponding to a hyperNode in CF. In this way, a complete execution is generated in a top-down fashion.

In the following, a complete execution of a workflow is formally defined as an outcome of the main program above.

Definition 5.8 (*Complete execution of a workflow*). Let $W = (N, CF, DF, IC, BC)$ be a workflow, and $CF = (N, E_{CF}, L, TC)$ be its control-flow, where CF itself is thought as a single node at an abstract level. A *Complete Execution of* W denoted as $CE = (N_{CE}, E_{CE})$ is a split-join hyperNodeDAG which can be generated through the Complete Execution Generation Algorithm (Algorithm 5.1).

Notice that there could be many complete executions that can be generated from the control-flow graph using Algorithm 5.1. The following example demonstrates the generation of a complete execution from a given control-flow.

Example 5.14. Consider the control-flow graph (CF) in figure 10. One of the complete executions that is generated from CF, e.g., $CE_1 = (N_{CE_1}, E_{CE_1})$, can be defined as follows: $N_{CE_1} = \{a, b, \{c, g, \{h, i, j, k, l\}, m\}, n\}$, and $E_{CE_1} = \{\langle a, b\rangle, \langle b, \varepsilon_3\rangle, \langle \varepsilon_3, n\rangle, \langle c, g\rangle, \langle g, \varepsilon_{33}\rangle, \langle \varepsilon_{33}, m\rangle, \langle h, i\rangle, \langle h, j\rangle, \langle h, k\rangle, \langle i, l\rangle, \langle j, l\rangle, \langle k, l\rangle\}$, where $\varepsilon_3 = \{c, g, \{h, i, j, k, l\}, m\}$, and $\varepsilon_{33} = \{h, i, j, k, l\}$.

As stated previously, basic constraints can be violated during a workflow execution; yet as one of the essential conditions to preserve them all complete executions must satisfy the criteria given in the following definition.

Definition 5.9 (*Validation complete control-flow*). Let $W = (N, CF, DF, IC, BC)$ be a workflow, and $BC = (V_{BC}, E_{BC}, CL_{BC}, VL_{BC})$ be its basic constraints graph. CF is a

Validation Complete Control-Flow if the following conditions hold for every complete execution $CE_i = (N_{CE_i}, E_{CE_i})$ of W:

(1) $(\forall\langle t, VS, \neg\mathcal{B}, and\rangle \in E_{BC}) : ((t \in base(N_{CE_i})) \Rightarrow (VS \subseteq base(N_{CE_i})))$.
(2) $(\forall\langle t, VS, \neg\mathcal{B}, or\rangle \in E_{BC}) : ((t \in base(N_{CE_i})) \Rightarrow (VS \cap base(N_{CE_i}) \neq \emptyset))$.

Conditions 1 and 2 state that if an activity (t) does not preserve a basic constraint (i.e., $\mathcal{O}_t \not\Rightarrow \mathcal{B}$), then every complete execution (CE_i) including this activity must contain activities which validate this basic constraint again (i.e., activities of the corresponding and-validating set or at least one activity of corresponding or-validating set). This property must be ensured by the workflow designers. Note that, if $Preserve(t, \mathcal{B}) = 1/2$ and invalidation of $\mathcal{B}$ is prevented by the preceding activities then $\mathcal{O}_t \Rightarrow \mathcal{B}$. In this case, t is not placed in BC.

The following example clarifies the definition above.

Example 5.15. *WarehouseAllocation* (Example 5.10), and *Billing* (Example 5.11) workflows have validation complete control-flows, since intuitively every complete execution of *WarehouseAllocation* workflow includes the activities in $\bigcup_{k=1}^{size(destList)} t_{UML(w_k)}$ if it includes $t_{RM(w_j)}$, and every complete execution of *Billing* workflow includes either t_{RS} or t_{MC} activity in the case $\mathcal{B}_2$ is falsified by t_{UUB}.

A workflow environment can be left in an incorrect state due to incorrect interleavings during the execution of activities of the same or different workflows even these individual workflows have validation complete control-flows. Furthermore inter-activity constraints can be invalidated and therefore input conditions of some activities may be false when they are executed. Both situations sacrifice the correctness of workflows. Before introducing a correctness notion, we provide a formal definition of concurrent execution of workflows, namely a complete execution history of workflows. To specify interleavings of workflows and their constituting activities clearly in this definition, time intervals are associated with them during execution.

Assuming a model consisting of a fully ordered set of points (instants) of time, a time interval TI is an ordered pair of points which represents its endpoints, i.e., $TI = [START(TI), END(TI)]$, where $START(TI)$ and $END(TI)$ denote the start-point and end-point of TI respectively. Two relations between the time intervals, namely *intersect* and *cover* are presented in Table 1. In this table, TI_i and TI_j represent two arbitrary time intervals. TI_i and TI_j intersect, which is denoted as $TI_i \cap TI_j \neq \emptyset$, if they have at least a common point of time. If TI_i covers TI_j this is denoted as $TI_i \supset TI_j$. These relations are used later in this section. More information about time intervals and relations between them can be found in [2].

Table 1. Relations defined on time intervals.

Relation	Condition
TI_i and TI_j intersect	$\neg(END(TI_i) < START(TI_j)) \wedge \neg(END(TI_j) < START(TI_i))$
TI_i covers TI_j	$(START(TI_i) < START(TI_j)) \wedge (END(TI_j) < END(TI_i))$

After introducing time intervals and required relations among them, the following definition of the complete execution history of workflows is presented.

Definition 5.10 (*Complete execution history of workflows*). A *Complete Execution History* $CH = (T_{CH}, E_{CH}, L_{CH})$ defined over a set of complete workflow executions $CE = \{CE_1, CE_2, \ldots, CE_n\}$, where $CE_1, CE_2, \ldots, CE_n$ are generated from control-flows of a set of workflows $W = \{W_1, W_2, \ldots, W_m\}$, is a labeled split-join hyperNodeDAG, where

- $T_{CH} = \bigcup_{i=1}^{n} N_{CE_i} \cup \{s_{CH}, j_{CH}\}$, where s_{CH} and j_{CH} denote the split and join nodes of CH respectively, and s_{CH}, j_{CH} are equal to f_{CH} and l_{CH} (first and last nodes of CH) respectively.
- $E_{CH} = (\bigcup_{i=1}^{n} E_{CE_i}) \cup (\bigcup_{i=1}^{n} \{\langle s_{CH}, N_{CE_i}\rangle, \langle N_{CE_i}, j_{CH}\rangle\})$.
- L_{CH} is the labels of the nodes, i.e., each node is labeled with its time interval TI. For a simple node S, $TI_S = [start(S), end(S)]$, where $start(S)$ and $end(S)$ denote the time instants when the activity is started and terminated respectively. For a hyperNode S, $TI_S = [min(START(TI_{S_i})), max(END(TI_{S_i}))]$, where S_i is a simple or a hyperNode of S (i.e., $S_i \subseteq S$).

In the following definition, a correctness criterion for a complete execution history of workflows is presented. In this definition, a correct complete execution history is characterized by referring to the properties of the workflow environment state at particular time instants. Intuitively, for an infinite sequence $\tau = 0, 1, 2, \ldots$ of time instants there is a corresponding sequence $St_0, St_1, St_2, \ldots$ of workflow environment states. The notation St_{event} is employed to denote a particular workflow environment state at the time instant with which the *event* is associated. For example, $St_{start(t)}$ denotes the state when activity t is started. If a constraint $\mathcal{F}$ holds at the time instant at which *event* occurs, this situation is represented as $St_{event} \models \mathcal{F}$.

Definition 5.11 (*Correct complete execution history*). A *Complete Execution History* $CH = (T_{CH}, E_{CH}, L_{CH})$ is *correct* if the following conditions hold:

(1) $(\forall t \in base(T_{CH})) : (St_{start(t)} \models \mathcal{I}_t)$.
(2) $(St_{start(f_{CH})} \models \mathcal{B}) \Rightarrow (St_{end(l_{CH})} \models \mathcal{B})$, where f_{CH} and l_{CH} are the first and last nodes of CH respectively, and $\mathcal{B} = \bigwedge_i \mathcal{B}_i$ where $\mathcal{B}_i \in B$, and B is basic constraints of the workflow system.

Condition 1 states that when an activity t involved in the history is started its input condition $\mathcal{I}_t$ should hold. Notice that since the individual activities are isolated, validity of their input conditions when they are started is a sufficient condition to execute them correctly. According to Condition 2, if the basic constraints of the workflow system are true when the history is started they should be true after the termination of the history.

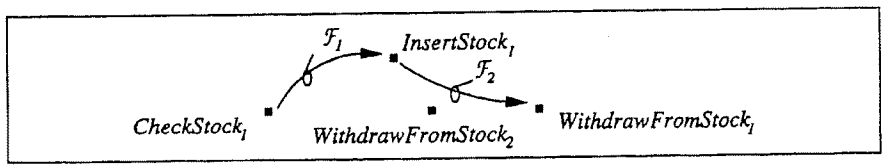
After defining a correctness notion for a complete execution history of workflows the ways correctness can be sacrificed are illustrated in the following paragraphs. If the execution of

activities of workflows are interleaved, correctness of a complete execution history can be violated in two ways:

- Input condition of an activity t may be false when t is executed (i.e., $St_{start(t)} \not\models \mathcal{I}_t$).
- Although basic constraints are true when the complete execution history is started, they may be false when it is terminated (i.e., $St_{execute(l_{CH})} \not\models \mathcal{B}$).

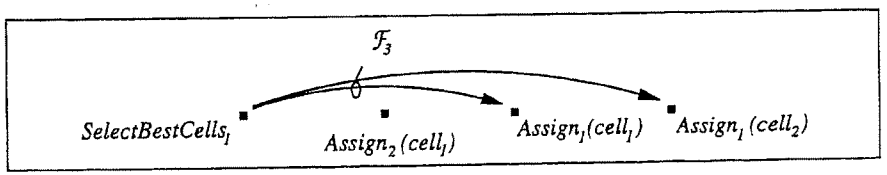
Input condition of an activity (Formula 11) can be violated in three ways: (1) An inter-activity constraint $\mathcal{F} \in C_{in}(t)$, or (2) a basic constraint $\mathcal{B} \in B(t)$, or (3) an extensional constraint $\mathcal{G} \in G(t)$ may not be true when t is executed. The following two examples demonstrate the first case.

Example 5.16. Consider the *CheckStock* (t_{CS}), *InsertStock* (t_{IS}), and *WithdrawFromStock* (t_{WFS}) activities, and the inter-activity constraints $\mathcal{F}_1 \equiv (quantity(m_i) \geq n)$, and $\mathcal{F}_2 \equiv (quantity(m_i) \geq required(m_i))$ given in Example 5.6. Remember that $\mathcal{F}_1 \in C_{[t_{CS}, t_{IS}]}$, and $\mathcal{F}_2 \in C_{[t_{IS}, t_{WFS}]}$. Since raw materials of type m_i may be withdrawn from the stock by the concurrently executing t_{WFS} activity of some other workflows, $\mathcal{F}_1$, and $\mathcal{F}_2$ may be invalidated between the t_{CS}, and t_{IS} activities, and corresponding t_{WFS} activity. This situation is depicted in the following:



Suppose that t_{CS_1} sees $n = 75$ raw materials in the stock and $required(m_i) = 125$; therefore 50 raw materials are ordered from vendors and inserted into stock through t_{IS_1} activity. After this, if a t_{WFS_2} activity of another instance of *OrderProcessing* workflow withdraws 30 raw materials of same type, input condition of t_{WFS_1} (i.e., $quantity(m_i) \geq 125$) is invalidated.

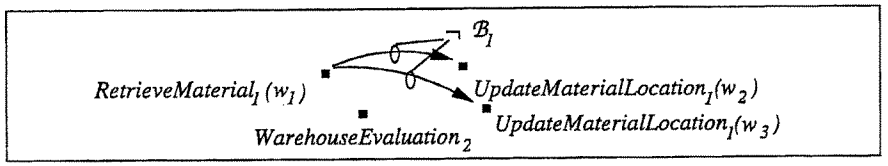
Example 5.17. Consider *SelectBestCells* (t_{SBC}) and *Assign* (t_A) activities, and the inter-activity constraint $\mathcal{F}_3 \equiv ((\forall cell_j \in (qualifiedCells)) : (rank(cell_i) \geq rank(cell_j)))$ defined in Example 5.7. Recall that $\mathcal{F}_3 \in C_{[t_{SBC}, t_{A(cell_i)}]}$. Since other t_A activities might concurrently assign a work to a preselected cells they can invalidate $\mathcal{F}_3$. This situation is depicted as follows:



Suppose that available cells are evaluated in t_{SBC_1}, and $cell_1$ and $cell_2$ are selected. If $t_{A_2(cell_1)}$ assigns a heavy work to $cell_1$, and degrades its previously assessed *rank*, $cell_1$ may become a worse selection for the assignment of the work in $t_{A_1(cell_1)}$. Thus input condition of $t_{A_1(cell_1)}$ may be invalid when it is executed.

The following example demonstrates a situation in which a basic constraint involved in the input condition of an activity is falsified.

Example 5.18. Consider Examples 5.5 and 5.10, and note that basic constraint $\mathcal{B}_1$ is false between *RetrieveMaterial*(w_j) (shortly $t_{RM(w_j)}$), and corresponding *UpdateMaterialLocation*(w_k) ($t_{UML(w_k)}$) activities for every $w_k \in destList$. If a *WarehouseEvaluation* (t_{WE}) activity is executed between these activities it executes incorrectly, since its input condition includes $\mathcal{B}_1$. This situation is demonstrated in the following:



Suppose that $t_{RM_1(w_1)}$ retrieves 1200 raw materials of type m_i from the stock of warehouse w_1 and these materials are distributed to stocks of warehouses w_2, and w_3 through $t_{UML_1(w_k)}$ activities. If t_{WE_2} activity is executed between them it misses the raw materials being transferred and an incorrect amount of raw material m_i is reported.

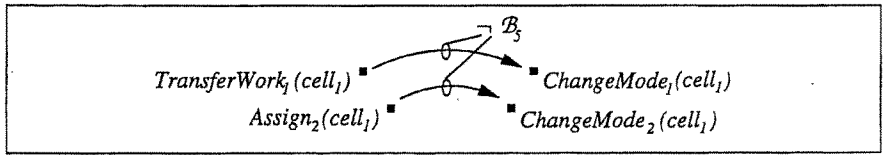
The preceding examples demonstrate the possible violations of input conditions. Now, we discuss the cases in which basic constraints may remain false after the termination of a complete execution history.

Note that validation completeness (Definition 5.9) is an essential requirement to preserve basic constraints in a complete execution history, thus if a basic constraint is invalidated by an activity it is revalidated by the execution of activities in its validating set. Yet to achieve this, the input conditions of activities in the validating set must hold when they are executed (Example 5.12). If input conditions of activities in a validating set are falsified, revalidation of a basic constraint fails. Thus, although workflows having validation complete control-flows are involved in a complete execution history, a workflow environment can be left in a state where basic constraints do not hold. The following example demonstrates this situation.

Example 5.19. Suppose that a basic constraint $\mathcal{B}_5$ is defined as follows:

$$\begin{aligned}\mathcal{B}_5 \equiv ((\forall cell_i \in cells) : (((&capacityMode(cell_i) = Normal)\\ &\Rightarrow (workload(cell_i) \leq \mathsf{C}_i)) \vee ((capacityMode(cell_i) = Max)\\ &\Rightarrow (\mathsf{C}_i < workload(cell_i) \leq \mathsf{MAX}_i))))\end{aligned} \tag{20}$$

The intuition behind this constraint is as follows: A manufacturing cell ($cell_i$) can work in normal (*Normal*) or maximum (*Max*) capacity modes. If $cell_i$ works in *Normal* mode, its workload should be equal or less than a predetermined upper limit C_i. In *Max* mode, its workload should be between C_i and MAX_i. Employing cells in *Normal* load is more desirable, and transferring a part of a workload to other available cells is possible. Consider the following executions of related activities:



Assume that $\mathsf{MAX}_1 = 500$, $\mathsf{C}_1 = 300$, and current workload of $cell_1$ is 400. *TransferWork*$_1$ ($cell_1$) (shortly $t_{TW_1(cell_1)}$) transfers a part of $cell_1$'s workload (i.e., 150) to other available cells. In this case, $\mathcal{B}_5$ is invalidated and *ChangeMode*$_1$($cell_1$) ($t_{CM_1(cell_1)}$) should be executed to change mode of $cell_1$ from *Max* to *Normal*. Notice that to guarantee validation of $\mathcal{B}_5$, inter-activity constraint

$$\mathcal{F}_4 \equiv (workload(cell_1) \leq \mathsf{C}_1) \tag{21}$$

must hold when $t_{CM_1(cell_1)}$ is executed. Thus, $cell_1$ works in *Normal* capacity mode with *workload* = 250, and therefore $\mathcal{B}_5$ is revalidated after the termination of $t_{CM_1(cell_1)}$. This situation is similar to one presented in Example 5.12. Consider the executions of activities which belong to another workflow instance. Suppose that *Assign*$_2$($cell_1$) ($t_{A_2(cell_1)}$) assigns a work to $cell_1$ in amount of 200, and therefore the resulting *workload* is 450. Since this workload requires *Max* capacity mode $t_{CM_2(cell_1)}$ is executed to validate $\mathcal{B}_5$, and *capacityMode*($cell_1$) is made *Max*. Note that the activities presented belong to workflows having validation complete control-flows. At the end of these executions, the resulting *capacityMode* is *Normal* and current *workload* is equal to 450. Thus $\mathcal{B}_5$ is still invalid. This is due to $t_{A_2(cell_1)}$ is invalidated $\mathcal{F}_4$ which is required for the correct execution of $t_{CM_1(cell_1)}$.

As discussed through the preceding examples, although individual activities of a workflow are executed in isolation, workflow correctness may be violated due to improper interleavings. Thus, proper concurrency control mechanisms are required to ensure correctness of a complete execution history. A concurrency control mechanism can guarantee that when t_j is executed $\mathcal{I}_{t_j}$ is true if it does not permit any activity that falsifies constraints in $C_{\{t_i,t_j\}}$ to be executed between t_i and t_j for different t_is. Furthermore, if a basic constraint involved in $\mathcal{I}_{t_j}$ is invalidated by a previously executed activity, execution of t_j should be delayed until this basic constraint is satisfied again by the activities of corresponding validating set. Revalidation of a basic constraint can be ensured by the validation completeness property, and guaranteeing correctness of input conditions of activities in a validating set.

Extensional constraints (i.e., $G(t_j)$) involved in the input condition of an activity may be falsified by the activities which are terminated even before the beginning of workflow in which t_j participates, and remain invalid for an uncertain time. Therefore, ensuring their validity like inter-activity or basic constraints through a concurrency control mechanism is not possible. A possible way to achieve this is that, a workflow designer places preceding activities in the control-flow to check these constraints, and if they evaluate to false either they are validated by proper activities or t_j is excluded from the execution history through conditional branches. Placing *CheckStock* and *InsertStock* activities before the *WithdrawFromStock* is an example to the first case. In this way, extensional constraints can be transformed to inter-activity constraints and their validity can be ensured like other constraints. If this design requirement is not taken into consideration by workflow designers, activity itself should verify extensional constraints, and if they evaluate to false, the activity should be removed from the execution history (e.g., by aborting it).

The essential design requirements which provide for the correctness of a complete execution history of workflows and hence must be ensured by the workflow designers can be summarized as follows: (1) Control-flow of workflows must be validation-complete; (2) proper inter-activity constraints must be introduced between the activities which invalidate and later revalidate a basic constraint; (3) extensional constraints must be transformed to inter-activity constraints, thus $G(t_j) = \emptyset$.

Theorem 5.1 provides the concurrency control requirements explained above in a formal manner. To specify the intervals where the basic constraints are (or may be) invalid, and where inter-activity constraints should be preserved at run-time in the theorem, time intervals (TI_E) are associated with the edges of a basic constraints graph (BC), and inter-activity constraints graph (IC) in the following:

- If E is an edge of an IC then, $TI_E = [START(TI_{source(E)}), END(TI_{sink(E)})]$, i.e., TI_E is denoted by the start of time interval associated with the source node and end of time interval associated with the sink node of E.
- If E is an edge of a BC, and $VL_{BC} = and$ then, $TI_E = [START(TI_{source(E)}), END(TI_{sink(E)})]$.
- If E is an edge of a BC, and $VL_{BC} = or$ then, $TI_E = [START(TI_{source(E)}), min(END(TI_{S_i}))]$, and $S_i \in sink(E)$, i.e., TI_E is denoted by the start of time interval associated with the source node, and minimum end-point of time intervals associated with the elements of the sink node of E. This is due to the fact that once an activity in $sink(E)$ is terminated, validity of a basic constraint is ensured.

Theorem 5.1 (Correctness of a complete execution history). *Let* $CH = (T_{CH}, E_{CH}, L_{CH})$ *be a complete execution history defined over a set of complete executions* $CE = \{CE_1, CE_2, \ldots, CE_n\}$, *where* $CE_1, CE_2, \ldots, CE_n$ *are generated from a set of workflows* $W = \{W_1, W_2, \ldots, W_m\}$ *having validation complete control-flows.* $W_i \in W$ *is represented as* $W_i = (N_i, CF_i, DF_i, IC_i, BC_i)$, *where* $IC_i = (V_{IC_i}, E_{IC_i}, L_{IC_i})$, *and* $BC_i = (V_{BC_i}, E_{BC_i}, CL_{BC_i}, VL_{BC_i})$. CH *is* correct *if the following conditions hold:*

(1) $St_{start(t_{CH})} \models \mathcal{B}$.

(2) $(\forall W_i \in W, \forall E \in E_{BC_i}, \forall t_x \in base(T_{CH})) : (TI_E \cap (\bigcup_x \{TI_{t_x} \mid \neg CL_{BC_i}(E) \in I_{t_x}\}) = \emptyset)$.

(3a) $(\forall W_i \in W, \forall E \in E_{IC_i}, \forall t_x \in base(T_{CH})) : (TI_E \cap (\bigcup_x \{TI_{t_x} \mid Preserve(t_x, L_{IC_i}(E)) = 0\}) = \emptyset)$.

(3b) $(\forall W_i \in W, \forall E \in E_{IC_i}, \forall t_x \in base(T_{CH})) : (((Preserve(t_x, L_{IC_i}(E)) = 1/2) \wedge (TI_E \cap TI_{t_x} \neq \emptyset)) \Rightarrow (St_{end(t_x)} \models L_{IC_i}(E)))$.

In the following, these conditions are explained to clarify them.

(1) Basic constraints (i.e., $\mathcal{B} \equiv \bigwedge_i \mathcal{B}_i$, where $\mathcal{B}_i \in B$) should hold when complete execution history (CH) is started (i.e., when its first activity, f_{CH}, is started).

(2) If $E = \langle t_j, VS = \{t_k, t_l, \ldots\}, CL_{BC_i}(E) = \neg\mathcal{B}_n, VL_{BC_i}(E) = and/or\rangle$ is an edge in BC_i (where BC_i a basic constraints graph of a workflow $W_i \in W$), and if $\neg CL_{BC_i}(E) = \mathcal{B}_n$ is involved in the input condition of another activity t_x (i.e., $\mathcal{B}_n \in I_{t_x}$), time intervals associated with E (TI_E) and t_x (TI_{t_x}) should not intersect.

(3a) If $E = \langle t_j, \{t_k, t_l, \ldots\}, L_{IC_i}(E) = \mathcal{F}\rangle$ is an edge in IC_i (where IC_i is an inter-activity constraints graph of a workflow $W_i \in W$), and if another activity t_x falsifies $\mathcal{F}$ (i.e., $Preserve(t_x, \mathcal{F}) = 0$), TI_E and TI_{t_x} should not intersect.

(3b) If $E = \langle t_j, \{t_k, t_l, \ldots\}, L_{IC_i}(E) = \mathcal{F}\rangle$ is an edge in IC_i, and t_x *may* falsify $\mathcal{F}$ (i.e., $Preserve(t_x, \mathcal{F}) = 1/2$), $\mathcal{F}$ should be still valid when t_x is terminated. Notice that, if t_x does not participate in CH (e.g., by removing it from CH), this condition automatically holds.

Proof: To prove this theorem, we show that if the conditions stated in Theorem 5.1 are true, the conditions in the definition of a correct complete execution history (i.e., Definition 5.11) hold.

(1) As a first step, it is proved that $(\forall t \in base(T_{CH})) : (St_{start(t)} \models \mathcal{I}_t)$ is true. Assume that $(\exists t_x \in base(T_{CH})) : (St_{start(t_x)} \not\models \mathcal{I}_{t_x})$. To achieve this, at least one of the conditions below should hold:

- $St_{start(t_x)} \not\models \mathcal{B}_i$, where $\mathcal{B}_i \in B(t_x)$.
- $St_{start(t_x)} \not\models \mathcal{F}_j$, where $\mathcal{F}_j \in C_{in}(t_x)$.
- $St_{start(t_x)} \not\models \mathcal{G}_k$, where $\mathcal{G}_k \in G(t_x)$.

Remember that the constraints constituting an input condition are the elements of $B(t_x) \cup C_{in}(t_x) \cup G(t_x)$ (Formula 11). Trivially, Condition 2 of Theorem 5.1 prevents first case; second case is not possible due to Conditions 3a and 3b. It is guaranteed that the last case does not occur by workflow design.

(2) In this step, it is proved that $(St_{start(f_{CH})} \models \mathcal{B}) \Rightarrow (St_{end(l_{CH})} \models \mathcal{B})$ holds. First part of the formula is true by assumption (i.e., Condition 1 of Theorem 5.1). Assume that $St_{end(l_{CH})} \not\models \mathcal{B}$; to achieve this $\mathcal{O}_{t_x} \not\Rightarrow \mathcal{B}$ should hold for a $t_x \in base(T_{CH})$. In this case, however, activities of an and/or-validating set are present in CH due to validation completeness property (Definition 5.9). It has been already proved that validity of input conditions of activities in a validating set are guaranteed. Thus, $\mathcal{B}$ is certainly validated prior to the termination of CH by these activities.

Thus, if the conditions of Theorem 5.1 are true, correctness of CH is guaranteed. □

6. Constraint based concurrency control (CBCC) mechanism

In this section, a *Constraint Based Concurrency Control* (*CBCC*) mechanism for workflows based on the correctness notion developed in Section 5 is proposed.

In Section 5 it is shown that, if the conditions of Theorem 5.1 hold, correctness of a complete execution history of workflows is guaranteed. Validity of these conditions can indeed be guaranteed through a *Constraint Based Concurrency Control* mechanism to control activity interleavings in such a way that inter-activity constraints are preserved and accesses to workflow environment on which the basic constraints do not hold are prevented. In this mechanism, activities acquire and release locks on inter-activity and basic constraints in two different modes, and certain inter-activity constraints are evaluated within an activity. To achieve this, CBCC mechanism employees three stages for the execution of an activity: (1) Locking stage before the actual execution of an activity; (2) Certification (evaluation) stage before the actual termination of an activity; (3) Lock releasing stage after an activity terminates. Activities acquire locks on the relevant constraints in the locking stage by issuing lock requests to CBCC mechanism. The lock compatibility table for inter-activity and basic constraints is given in Table 2. "Y" means that the locks do not conflict and "N" means the locks conflict.

An inter-activity constraint $\mathcal{F}$ can be locked by an activity t_x in one of the following modes:

- **Shared:** This mode of lock is acquired when t_x intends to preserve $\mathcal{F}$ until a set of other activities terminate, i.e., $\mathcal{F} \in C_{out}(t_x)$.
- **Exclusive:** This mode is used when t_x falsifies $\mathcal{F}$, i.e., $Preserve(t_x, \mathcal{F}) = 0$. All inter-activity constraints in a workflow management system which are falsified by t_x constitute the set $F(t_x)$. Note that not only inter-activity constraints within a workflow in which t_x resides, but also all inter-activity constraints of other workflows are considered for this set.

If $\mathcal{F}$ is to be preserved in the interval between activity t_j and a set of activities $\{t_k, t_l, \ldots\}$, and if another activity t_x that falls in this interval falsifies $\mathcal{F}$, t_x should be delayed until $\mathcal{F}$ is unlocked by the every activity in $\{t_k, t_l, \ldots\}$. Therefore, the shared lock taken by t_j conflicts with the exclusive lock taken by t_x, as indicated in Table 2. Furthermore if $\mathcal{F}$ is to be preserved in the interval between activities t_j and $\{t_k, t_l, \ldots\}$, and again $\mathcal{F}$ is to be preserved in another interval between t_m and $\{t_n, t_o, \ldots\}$, both t_j and t_m lock $\mathcal{F}$ in shared mode and clearly there is no need for these shared locks to be in conflict, as indicated in Table 2. Note that we use the term "exclusive lock" differently than its conventional meaning in that, two exclusive locks on the same constraint do not conflict with each other in our approach as opposed to traditional exclusive locks.

Table 2. The lock compatibility table for inter-activity and basic constraints.

Mode Requested	Existing: Shared	Existing: Exclusive
Shared	Y	N
Exclusive	N	Y

It should be noted that some of the inter-activity constraints *may be* falsified by t_x, i.e., $Preserve(t_x, \mathcal{F}) = 1/2$, which constitute the set $LF(t_x)$. For the activities that may falsify inter-activity constraints, we prefer to use an optimistic scheme rather than locking with the intention of increasing the performance, since there is a probability that the activity will not falsify these constraints. If a constraint in this set is already locked in shared mode to be maintained when t_x is executed, this constraint is evaluated in the certification stage and if it evaluates to false, t_x is rolled back and resubmitted to workflow management system.

A basic constraint $\mathcal{B}$ can be locked by t_x in one of the following modes:

- **Shared:** If t_x requires the correctness of $\mathcal{B}$, i.e., $\mathcal{B} \in B(t_x)$, a shared lock is acquired.
- **Exclusive:** If t_x invalidates (or may invalidate) $\mathcal{B}$, i.e., $\mathcal{B} \in (\bigcup_{VS} SB_{(t_x, VS, and/or)})$, an exclusive lock is required.

An activity t_x (may) falsify a basic constraint $\mathcal{B}$ to be revalidated by the activities of and/or-validating sets as explained in Section 5. Therefore the activities that require the correctness of $\mathcal{B}$ in this interval should not be allowed to execute. For this reason, t_x obtains an exclusive lock on $\mathcal{B}$. On the other hand the activity that requires the correctness of $\mathcal{B}$ acquires a shared lock. The shared lock conflicts with the exclusive lock as indicated in Table 2. It is clear that the activities that require correctness of $\mathcal{B}$ do not conflict with each other.

6.1. CBCC algorithms

In this section, the algorithms employed by CBCC mechanism are described. In these algorithms, data structures *IC*, *BC* for every workflow, and $B(t_x)$, $F(t_x)$, $LF(t_x)$ for every activity are required. A *Constraint Editor* in conjunction with a first-order constraint specification language [12, 15] can be used by an administrator and/or workflow designers to define these data structures.

6.1.1. Algorithm for activity start. Any activity t_x needs an exclusive lock for every inter-activity constraint it falsifies to start (*Steps* 1 and 2 of *Algorithm* 6.1). This is possible only when there is no other activity that has a shared lock on $\mathcal{F}$; in other words no other activity wants to preserve $\mathcal{F}$. Furthermore, t_x also needs to acquire shared locks for all the basic constraints involved in its input condition (i.e., $B(t_x)$) (*Steps* 3 and 4). A lock for a constraint $\mathcal{B}$ in $B(t_x)$ is granted to t_x if there is no invalidating activity that has an exclusive lock on $\mathcal{B}$. After this step, every inter-activity constraint emanating from t_x in the inter-activity constraints graph (*IC*) (i.e., elements of $C_{out}(t_x)$) are locked in the shared mode in *Steps* 5 and 6. t_x can acquire a shared lock on $\mathcal{F} \in C_{out}(t_x)$ if no other invalidating activity for $\mathcal{F}$ has an exclusive lock on $\mathcal{F}$. Recall that $\mathcal{F}$ may be incident to more than one activity, and these activities are grouped into a hyperSet $S_{(t_x, \mathcal{F})}$. This is represented by the edge $\langle t_x, S_{(t_x, \mathcal{F})}, \mathcal{F} \rangle$ in *IC*. Since $\mathcal{F}$ should be preserved until the termination of

all the activities in the hyperSet $S_{(t_x,\mathcal{F})}$, it is necessary to obtain a shared lock for each of the activities in this set, i.e., $size(S_{(t_x,\mathcal{F})})$ locks are acquired. A conflicting lock can then only be allowed when all these locks are released. In *Steps* 7–10, exclusive locks are acquired on the basic constraints which are invalidated by t_x which is only possible if there are no shared locks on $\mathcal{B}$. That is, since t_x is invalidating $\mathcal{B}$, there should not exist any activity that requires the correctness of $\mathcal{B}$. If *VS* is an and-validating set for $\mathcal{B}$ and if it contains more than one activity, t_x acquires an exclusive lock on $\mathcal{B}$ for each activity of *VS*, that is the number of locks acquired is $size(VS)$. If *VS* is an or-validating set, t_x acquires a single lock since the termination of the first activity of *VS* guarantees validity of $\mathcal{B}$.

Inter-activity constraints which *may* be falsified by t_x, i.e., $LF(t_x)$ are handled in an optimistic manner. Note that all the constraints in $LF(t_x)$ may not be active, that is, it may be the case that for some constraints in $LF(t_x)$, there is no activity requiring these constraints to hold. We include all the active constraints in *ActiveICS* set and all the constraints in this set are already locked in the shared mode. The intersection of $LF(t_x)$ and *ActiveICS* sets gives us the set of constraints denoted as $ALF(t_x)$, that are both active when t_x has started and also has to be validated when t_x terminates (*Step* 11). Since new shared locks can be acquired on the elements of $LF(t_x) - ActiveICS$ by other activities before the activity terminates, constraints in $PLF(t_x) = LF(t_x) - ActiveICS$ (i.e., non-active constraints which are in $LF(t_x)$) are locked in exclusive mode. Furthermore, operations in *Step* 11 are executed atomically (i.e., in a critical section). In this way, further constraints that may be falsified by t_x are prevented from becoming active after the set of constraints that will be validated are determined.

Algorithm 6.1 (Algorithm for activity start).

begin
1. **for** *every* $\mathcal{F} \in F(t_x)$ **do**
2. $\quad ExclusiveLock(\mathcal{F})$;
3. **for** *every* $\mathcal{B} \in B(t_x)$ **do**
4. $\quad SharedLock(\mathcal{B})$;
5. **for** *every* $\mathcal{F} \in C_{out}(t_x)$ **do**
6. $\quad SharedLock(\mathcal{F})$ *with Counter* $= size(S_{(t_x,\mathcal{F})})$;
7. **for** *every* $\mathcal{B} \in (\bigcup_{VS} SB_{\{t_x,VS,and\}})$ **do**
8. $\quad ExclusiveLock(\mathcal{B})$ *with Counter* $= size(VS)$;
9. **for** *every* $\mathcal{B} \in (\bigcup_{VS} SB_{\{t_x,VS,or\}})$ **do**
10. $\quad ExclusiveLock(\mathcal{B})$;
11. $\left[\begin{array}{l} ALF(t_x) \leftarrow (LF(t_x) \cap ActiveICS); \\ PLF(t_x) \leftarrow (LF(t_x) - ActiveICS); \\ \textbf{for } every\ \mathcal{F} \in PLF(t_x)\ \textbf{do} \\ \quad ExclusiveLock(\mathcal{F}) \end{array}\right]$

end

After successfully acquiring all the necessary locks as indicated in the *Algorithm* 6.1, an activity can be scheduled for execution.

6.1.2. Algorithm for activity end. An activity terminates when all of its operations are complete. But prior to termination, an evaluation algorithm (*Algorithm* 6.2) is executed to check whether an active inter-activity constraint is falsified by the execution of this activity. This is achieved in *Step* 1 by evaluating the constraints in $ALF(t_x)$ in parallel by the routine *EvalInParallel*; once a constraint evaluates to *false*, *EvalInParallel* terminates immediately and returns *false*. In this case, the activity t_x is rolled backed and resubmitted to workflow management system. Note that all the locks acquired by t_x should be released. If $ALF(t_x)$ is empty, *Algorithm* 6.2 is not executed.

Algorithm 6.2 (Algorithm for activity end).

begin
1. **if** $(EvalInParallel(ALF(t_x)) = false)$ **then**
2. $\quad Rollback(t_x), Resubmit(t_x)$

end

6.1.3. Algorithm for activity post-processing. After an activity t_x is terminated, all locks acquired by t_x on the constraints in $PLF(t_x)$, $F(t_x)$, and $B(t_x)$ are released in *Steps* 1 and 2, 3 and 4, and 5 and 6 of *Algorithm* 6.3 respectively. Inter-activity constraints incident to t_x (i.e., $C_{in}(t_x)$) which are locked by other activities are released in *Steps* 7 and 8. If t_x is in an and-validating set (*VS*) of a basic constraint $\mathcal{B}$, one of the previously acquired exclusive locks by the invalidating activity of $\mathcal{B}$ is released in *Steps* 9 and 10. If t_x is the first terminating activity of an or-validating set, a corresponding lock is released *Steps* 11 and 12.

Algorithm 6.3 (Algorithm for activity post-processing).

begin
1. **for** *every* $\mathcal{F} \in PLF(t_x)$ **do**
2. $\quad Unlock(\mathcal{F})$;
3. **for** *every* $\mathcal{F} \in F(t_x)$ **do**
4. $\quad Unlock(\mathcal{F})$;
5. **for** *every* $\mathcal{B} \in B(t_x)$ **do**
6. $\quad Unlock(\mathcal{B})$;
7. **for** *every* $\mathcal{F} \in C_{in}(t_x)$ **do**
8. $\quad Unlock(\mathcal{F})$;
9. **for** *every* $\mathcal{B} \in (\bigcup_{t_i} SB_{\{t_i,VS,and\}})$ *where* $t_x \in VS$ *or* $t_x = VS$ **do**
10. $\quad Unlock(\mathcal{B})$;
11. **for** *every* $\mathcal{B} \in (\bigcup_{t_i} SB_{\{t_i,VS,or\}})$ *where* $t_x = first(VS)$ **do**
12. $\quad Unlock(\mathcal{B})$

end

6.2. Correctness of the CBCC mechanism

To prove that a complete execution history (*CH*) generated by CBCC mechanism is correct we show that the conditions of Theorem 5.1 hold for *CH*. The following properties about

time intervals are used in the proof. Note that $\supset$ and $\cap$ denote cover and intersect relations between the time intervals respectively.

- $((TI_i \supset TI_j) \wedge (TI_j \cap TI_k \neq \emptyset)) \Rightarrow (TI_i \cap TI_k \neq \emptyset)$.
- $((TI_i \supset TI_j) \wedge (TI_i \cap TI_k = \emptyset)) \Rightarrow (TI_j \cap TI_k = \emptyset)$.

Theorem 6.1. *Any complete execution history (CH) generated by CBCC mechanism is correct.*

Proof:

(1) Condition 1 of Theorem 5.1 holds due to the assumption.

(2) Assume that Condition 2 of Theorem 5.1 does not hold; hence $TI_E \cap TI_{t_x} \neq \emptyset$ in CH for an edge $E = \langle t_j, VS = \{t_k, t_l, \ldots\}, \neg\mathcal{B}_n, and/or\rangle$ in BC_i, and an activity t_x where $\mathcal{B}_n \in B(t_x) \subseteq I_{t_x}$. The interval between the time when an exclusive lock on $\mathcal{B}_n$ is acquired with counter by t_j and the time when the last of these locks are released is denoted as $TI_E^{XL(\mathcal{B}_n)}$ in the case where VS is an and-validating set. Same notation is used to denote the interval between the time instances where a single lock is acquired by t_j and released by the first activity of an or-validating set VS. Similarly, the interval between the time when a shared lock is acquired and released on $\mathcal{B}_n$ by t_x is denoted as $TI_{t_x}^{SL(\mathcal{B}_n)}$. Since activities acquire locks before they start and release after they complete, $TI_E^{XL(\mathcal{B}_n)} \supset TI_E$ and $TI_{t_x}^{SL(\mathcal{B}_n)} \supset TI_{t_x}$. Since exclusive and shared locks on a basic constraint conflict, it is guaranteed that $TI_E^{XL(\mathcal{B}_n)} \cap TI_{t_x}^{SL(\mathcal{B}_n)} = \emptyset$. Yet, due to first property above $((TI_E^{XL(\mathcal{B}_n)} \supset TI_E) \wedge (TI_E \cap TI_{t_x} \neq \emptyset)) \Rightarrow (TI_E^{XL(\mathcal{B}_n)} \cap TI_{t_x} \neq \emptyset)$. Furthermore, according to second property, $((TI_{t_x}^{SL(\mathcal{B}_n)} \supset TI_{t_x}) \wedge (TI_{t_x}^{SL(\mathcal{B}_n)} \cap TI_E^{XL(\mathcal{B}_n)} = \emptyset)) \Rightarrow (TI_{t_x} \cap TI_E^{XL(\mathcal{B}_n)} = \emptyset)$. Observe that the right hand sides of two formulas contradict each other; hence our presumption is false and Condition 2 of Theorem 5.1 holds.

(3a) We start with proving that if $Preserve(t_x, \mathcal{F}) = 0$ then $TI_E \cap TI_{t_x} = \emptyset$ is guaranteed in CH for an edge $E = \langle t_j, \{t_k, t_l, \ldots\}, \mathcal{F}\rangle$ in IC_i. We denote the interval between the time when a shared lock on $\mathcal{F}$ is acquired with counter by t_j and the time when the last of these locks are released as $TI_E^{SL(\mathcal{F})}$. Similarly, the interval between the time when an exclusive lock is acquired and released on $\mathcal{F}$ by t_x is denoted as $TI_{t_x}^{XL(\mathcal{F})}$. Again, $TI_E^{SL(\mathcal{F})} \supset TI_E$ and $TI_{t_x}^{XL(\mathcal{F})} \supset TI_{t_x}$. Since exclusive and shared locks on an inter-activity constraint conflict, it is ensured that $TI_E^{SL(\mathcal{F})} \cap TI_{t_x}^{XL(\mathcal{F})} = \emptyset$. With the similar observations as in Condition 2 of this proof, Condition 3a of Theorem 5.1 holds.

(3b) We conclude with proving that if $Preserve(t_x, \mathcal{F}) = 1/2$, $TI_E \cap TI_{t_x} \neq \emptyset$ implies $\mathcal{F}$ holds after t_x is terminated. Depending on the execution sequences of t_j and t_x two possibilities can occur:

- t_j acquires a shared lock on $\mathcal{F}$ before t_x acquires an exclusive lock on $\mathcal{F}$: $\mathcal{F}$ is certainly logged into $ALF(t_x)$ and if t_x falsifies $\mathcal{F}$, $EvalInParallel(ALF(t_x))$ returns *false* and t_x is removed from CH (i.e., rolled backed); hence $TI_E \cap TI_{t_x} = \emptyset$.
- t_x acquires an exclusive lock on $\mathcal{F}$ before t_j acquires a shared lock on $\mathcal{F}$: t_j can not lock $\mathcal{F}$ in shared mode after *Step* 11 of *Algoritm* 6.1 and before t_x terminates, since t_x already locked $\mathcal{F}$ in exclusive mode in *Step* 11. Hence $TI_E \cap TI_{t_x} = \emptyset$.

Thus, a complete execution history generated by CBCC mechanism is correct. □

6.3. *Discussion*

There are several alternatives to implement a constraint based concurrency control mechanism. In the following, some of these alternatives are discussed:

- *Conservative.* In this approach, activities that are certainly or likely to falsify basic and inter-activity constraints are determined in advance (i.e., in design-time), and possible invalidations of inter-activity constraints and accesses to states on which the basic constraints do not (or may not) hold are prevented conservatively. For example, proposed CBCC mechanism can be classified into this category if activities try to acquire locks on the inter-activity constraints which they may falsify in addition to constraints which they certainly falsify in *Steps* 1 and 2 of *Algorithm* 6.1. Also *Step* 11 of *Algorithm* 6.1, and *Algorithm* 6.2 become unnecessary in this case. Since this conservative technique is based solely on locking, we call it as the *Constraint Locking Concurrency Control* (*CLCC*) mechanism. In CLCC mechanism, constraints themselves are no longer necessary, but can be represented through some simple data items just for locking purposes. It should also be noted that, if such a technique is not implemented in a workflow system, it is possible to acquire locks manually on virtual data items using the same principles.
- *Optimistic.* In this approach, activities validate their input conditions. This requires additional operations for the verification of these conditions. Optimistic technique is very similar to concurrency control mechanism of ConTract model [56]; however the input conditions we check are well-defined interms of inter-activity and basic constraints. If input condition of an activity evaluates to false, a conflict resolution algorithm can be executed to correct the input condition violation or to relax the requirements in the input condition. An inevitable result may be abortion of the activity and compensation of some previously terminated activities.
- *Dynamic-conservative.* The approach employed by the CBCC mechanism can be classified into this category.

In the optimistic technique, if conflict resolution algorithm requires rollback of the activity this may cause (possibly cascading) compensation of previously terminated activities which may be a very costly process [40, 48]. In addition, overhead of validation of every input condition should not be ignored. CLCC and CBCC techniques guarantee that input condition of an activity is true when it is executed; thus neither input condition validation nor compensation of other activities to resolve conflicts are required in these techniques. In addition, CBCC mechanism provides some activities to be executed and terminated if they pass certification process although these activities and consequently successor activities would be blocked by the CLCC mechanism. Furthermore, in the optimistic technique it is necessary to check the constraints themselves; however in CLCC mechanism these constraints can be represented by some simple data items just for locking purposes. In CBCC mechanism on the other hand, only the inter-activity constraints which may be falsified

by the activities are needed in the validation phase. In Section 6.4, a comparison of the performance characteristics of these techniques is provided.

It should be noted that, proposed CBCC and CLCC mechanisms may result in deadlocks like any other locking-based concurrency control mechanism, since activities may be blocked indefinitely. Therefore, special algorithms are required to handle deadlocks. There are three well known types of methods for handling deadlocks: prevention, avoidance, and detection and resolution [46]. We have developed a deadlock avoidance technique for CBCC and CLCC mechanisms in which potential deadlock situations are detected in advance (i.e., in design-time) and it is ensured that they will not occur at run-time by imposing additional restrictions on the interleavings of activities. Since concurrency control dependencies among activities are known in advance, possible deadlock situations can be detected in design-time in CBCC and CLCC mechanisms. Detailed explanation and formal foundation of this approach are presented in [6] due to space limitations.

6.4. *Performance analysis*

In this section, a performance comparison of the CBCC, CLCC mechanisms and optimistic technique which is similar to concurrency control mechanism of ConTract model [56] is given. The simulation is realized in GPSS [53]. In the experiments, average response time of a workflow instance (*avgResTime*) is measured by averaging response times of 10 workflow instances. Response time is defined as the time between the generation and termination of a workflow instance.

In the simulation, there are a total of 10 different basic and inter-activity constraints in the system. It should be noted that, the total number of constraints are kept small so that the possibility of conflicts among activities is high. In this way, the performances of the methods can be observed in a very high conflict case. For each activity, the number of constraints that should be considered (i.e., locked or evaluated) is randomly chosen from the interval [0–*maxCons*] where *maxCons* denotes the maximum number of constraints per activity and is given a priori. In the CLCC mechanism, each activity tries to obtain a lock on all of its constraints. Note that, some of the constraints which *may* be falsified by an activity are evaluated at the activity end instead of being locked in the CBCC mechanism. The evaluation cost per constraint is taken as constant for simplicity (i.e., 5 simulation time units). If a constraint evaluates to false the activity is aborted and restarted later. In the optimistic technique, the constraints are evaluated when the activity starts and once a constraint evaluates to false the activity is aborted and preceding activities are compensated. The result of the evaluation is randomly determined as true or false with the probability of 70 and 30% respectively. It should be noted that this fraction favors the optimistic technique rather than the CBCC mechanism, because in the CBCC mechanism a small fraction of constraints goes through the validation as opposed to all constraints in the optimistic method. Also in favor of the optimistic technique, the compensation cost is chosen as close to the maximum duration of just one activity, i.e., 50 simulation time units, although in reality this cost is much higher since compensation of more than one activity is more probable.

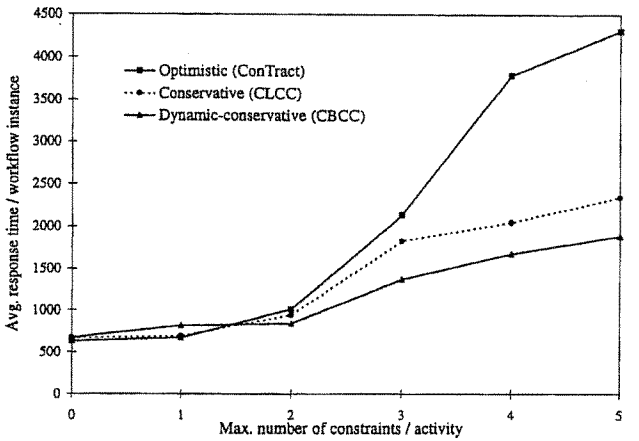


Figure 14. Average response times for different maximum number of constraints per activity.

The graph in figure 14 shows the average workflow instance response times (*avgResTime*) of three techniques for different maximum number of constraints per activity (*maxCons*). The experiment results can be summarized as follows:

All techniques provide their best *avgResTimes* when *maxCons* is small, i.e., in [0–2]. This is expected since when $maxCons < 2$ the probability of conflicts among activities is low, and consequently the number of blocked or compensated activities is small.

When $maxCons \geq 2$, CBCC and CLCC techniques provide better *avgResTimes* than optimistic technique. For example, when *maxCons* is equal to half of the total number of constraints in the system (e.g., around 5), *avgResTime* provided by the optimistic technique becomes worse than two times of *avgResTime* provided by CBCC mechanism, i.e., 1884 vs. 4306 simulation time units. This is due to fact that, the number of compensated activities increases in the optimistic technique with the increasing number of constraints (*maxCons*) which implies higher rate of conflicts. In CBCC mechanism, however, abortion of an activity does not lead to compensation of previous activities, only the activity itself is retried later.

When $maxCons = 2$, CBCC mechanism starts to perform better than CLCC mechanism. For example, when $maxCons \geq 3$, CBCC mechanism provides approximately 25% faster *avgResTime* than CLCC technique. Since not all the constraints are locked in the CBCC mechanism, the probability of delays due to locking is lower than that of CLCC mechanism. This difference becomes more visible when *maxCons* is larger.

Performance results presented indicate that the CBCC mechanism results in lower average workflow instance response times in almost all cases except when maximum number of constraints that should be considered per activity (*maxCons*) is very small (e.g., 1) or such a constraint does not exist. If *maxCons* is small, *avgResTimes* provided by the compared techniques are almost the same.

After observing that the performance of the optimistic technique is not good in a high conflict case, additional experiments are conducted to compare the performances of CBCC

and CLCC techniques for different evaluation costs. These experiment results are presented in [6].

7. Conclusions

Concurrency control aspects of workflow systems is addressed in this work, which is very important for some workflow applications where mission critical operations require the consistent view of the execution environment [21].

The fundamental issue of correctness criterion specific to workflow systems is defined through inter-activity constraints and basic constraints by using the semantic workflow information available at design-time. A concurrency control technique, namely Constraint Based Concurrency Control (CBCC) mechanism, based on this criterion is defined which uses the concept of locking in conjunction with validation with a fundamental difference from the database locking: the constraints rather than data items are locked. We have shown that, with a proper constraint locking and validation mechanism, the inter-activity constraints that should remain valid are preserved, and the activities that need basic constraints to hold are prevented from executing in the intervals where these constraints do not hold. It is also possible to use a more conservative approach in which the activities acquire locks instead of going through a validation phase. We call this technique as Constraint Locking Concurrency Control (CLCC) mechanism. These techniques are simple to implement, and the performance analysis indicates that the suggested techniques have better performance than an optimistic approach based on the constraints (similar to ConTract [56]).

Providing flexibility and preserving correctness are somewhat conflicting aims. In the suggested techniques a workflow designer introduces constraints to provide for the correctness of workflows. However when the correctness is not an issues for parts of a workflow, it is possible to have a more flexible system. When a workflow designer does not require the correctness to be preserved, some of the constraints may not be enforced. In this respect, it is possible to apply an isolation mechanism similar to isolation levels in databases [31] by allowing the workflow designer to customize the constraints graphs according to the correctness requirements of workflow application. For these reasons, we believe that the CBCC and CLCC techniques have practical importance.

References

1. D. Agrawal, A.E. Abbadi, and A.K. Singh, "Consistency and orderability: Semantics-based correctness criteria for databases," ACM TODS, vol. 18, no. 3, September 1993.
2. J. Allen, "Maintaining knowledge about temporal intervals," Com. ACM, vol. 26, no. 11, November 1983.
3. G. Alonso, D. Agrawal, and A.E. Abbadi, "Process synchronization in workflow management systems," 8th IEEE Symposium on Parallel and Distributed Processing, 1996.
4. G. Alonso and H.-J. Schek, "Research issues in large workflow management systems," in Proc. of NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, A. Sheth (Ed.), Athens, Georgia, May 1996.
5. P. Ammann, S. Jajodia, and I. Ray, "Applying formal methods to semantic-based decomposition of transactions," ACM TODS, vol. 22, no. 2, June 1997.
6. I.B. Arpinar, "Formalization of workflows and correctness issues in the presence of concurrency," Ph.D. thesis, Dept. of Computer Eng., Middle East Tech. Univ., November, 1998.
7. I.B. Arpinar, S. (Nural) Arpinar, U. Halici, and A. Dogac, "Correctness of workflows in the presence of concurrency," Next Generation Info. Tech. and Sys., Israel, July 1997.

8. P.A. Attie, M.P. Singh, E. Emerson, A. Sheth, and M. Rusinkiewicz, "Scheduling workflows by enforcing intertask dependencies," Dist. Sys. Engineering, vol. 3, no. 4, pp. 222–238, December 1996.
9. P.A. Attie, M.P. Singh, A. Sheth, and M. Rusinkiewicz, "Specifying and enforcing intertask dependencies," in Proc. of the 19th Intl. Conf. on VLDB, September 1993.
10. B. Badrinath and K. Ramamritham, "Semantics-based concurrency control: Beyond commutativity," in Proc. of Intl. Conf. on Data Engineering, February 1987.
11. C. Beeri, P.A. Bernstein, and N. Goodman, "A model for concurrency in nested transaction systems," Journal of the ACM, vol. 36, no. 2, 1989.
12. M. Benedikt, T. Griffin, and L. Libkin, "Verifiable properties of database transactions," ACM PODS 1996, Montreal, Canada.
13. A.J. Bernstein and P.M. Lewis, "Transaction decomposition using transaction semantics," Distributed and Parallel Databases, vol. 4, pp. 25–47, 1996.
14. Y. Breitbart, A. Deacon, H.J. Schek, A. Sheth, and G. Weikum, "Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows," ACM SIGMOD Record, vol. 22, no. 3, September 1993.
15. S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca, "Automatic generation of production rules for integrity maintenance," ACM TODS, vol. 19, no. 3, September 1994.
16. E.W. Dijkstra, A Discipline of Programming, Prentice-Hall: Englewood Cliffs, NJ, 1976.
17. A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, I.B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, and M. Altinel, Design and Implementation of a Distributed Workflow Management System: METUFlow, in Advances in Workflow Management Systems and Interoperability, Springer Verlag, 1998.
18. A. Dogac, L. Kalinichenko, M.T. Ozsu, and A. Sheth (Eds.), Advances in Workflow Management Systems and Interoperability, Springer Verlag, 1998.
19. J.A. Ellis and G.J. Nutt, "Modeling and enactment of workflow systems," 14th Intl. Conf. on Application and Theory of Petri Nets, 1993.
20. A.K. Elmagarmid (Ed.), Database Transaction Models for Advanced Applications, Morgan Kaufmann Publishers: San Mateo, CA, 1992.
21. A. Elmagarmid and W. Du, Workflow Management: State of the Art vs. State of the Market, in Advances in Workflow Management Systems and Interoperability, Springer Verlag, 1998.
22. E.A. Emerson, "Temporal and modal logic," in Handbook of Theoretical Computer Science, J. van Leeuwen (Ed.), Elseiver 1990.
23. A. Farrag and M.T. Ozsu, "Using semantic knowledge of transactions to increase concurrency," ACM TODS, vol. 14, no. 4, December 1989.
24. M. Fitting, First Order Logic and Automated Theorem Proving, Springer Verlag: NY, 1990.
25. H. Garcia-Molina, "Using semantic knowledge for transaction processing in a distributed database," ACM TODS, vol. 8, no. 2, June 1983.
26. H.J. Genrich, "Predicate/transition nets," in Advances in Petri Nets, Springer, 1986, p. 254.
27. D. Georgakopoulos, M. Hornick, and F. Manola, "Customizing transaction models and mechanisms in a programmable environment supporting reliable workflow automation," IEEE Trans. on Knowledge and Data Eng., 1995.
28. D. Georgakopoulos, M. Hornick, and A.P. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," Distributed and Parallel Databases, vol. 3, pp. 119–153, 1995.
29. D. Georgakopoulos, M. Rusinkiewicz, and A.P. Sheth, "Using tickets to enforce the serializability of multidatabase transactions," IEEE TKDE, vol. 6, no. 1, 1994.
30. E. Gokkoca, M. Altinel, I. Cingil, N. Tatbul, P. Koksal, and A. Dogac, "Design and implementation of a distributed workflow enactment service," in Proc. of Intl. Conf. on Cooperative Information Systems, Charleston, USA, June 1997.
31. J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers: San Mateo, CA, 1993.
32. U. Halici, I.B. Arpinar, and A. Dogac, "Serializability of nested transactions in multidatabases," Intl. Conf. on Database Theory (ICDT '97), Greece, January 1997.
33. T. Harder, "Handling hot spot data in DB-sharing systems," Info. Sys., vol. 13, no. 2, 1988.

34. D. Harel et al., "Statemate: A working environment for the development of complex reactive systems," IEEE Transactions on Software Engineering, vol. 16, no. 4, April 1990.
35. M.P. Herlihy and W.E. Weihl, "Hybrid concurrency control for abstract data types," J. Comput. Syst. Sci., vol. 43, no.1, August 1991.
36. C. Hoare, "An axiomatic basis for computer programming," Com. ACM, vol. 12, no. 10, October 1969.
37. D. Hollinsworth, "The workflow reference model," Technical Report TC00-1003, Workflow Management Coalition, December 1994. Accessible via: http://www.aiai.ed.ac.uk/WfMC/
38. P. Karagoz, S. Arpinar, P. Koksal, N. Tatbul, E. Gokkoca, and A. Dogac, "Task handling in workflow management systems," Intl. Workshop on Issues and Applications of Database Technology, Berlin, June 1998.
39. P. Koksal, S. Arpinar, and A. Dogac, "Workflow history management," ACM Sigmod Record, vol. 27, no. 1, March 1998.
40. H.F. Korth, E. Levy, and A. Siberschatz, "A formal approach to recovery by compensating transactions," in Proc. of the 16th VLDB Conf., Brisbane, Australia, 1990.
41. H.F. Korth and G. Speegle, "Formal aspects of concurrency control in long-duration transaction systems using the NT/PV model," ACM TODS, vol. 19, no. 3, 1994.
42. N. Krishnakumar and A. Sheth, "Managing heterogeneous multi-system tasks to support enterprise-wide operations," Distributed and Parallel Databases, vol. 3, no. 2, pp. 155–186, April 1995.
43. N.A. Lynch, "Multilevel atomicity: A new correctness for database concurrency control," ACM TODS, vol. 8, no. 4, pp. 484–502, December 1983.
44. J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh, "WebWork: METEOR$_2$'s web-based workflow management system," Journal of Intel. Info. Sys., vol. 10, no. 2, March 1998.
45. P. Muth, D. Wodtke, J. WeiBenfels, G. Weikum, and A.K. Dittrich, "Enterprise-wide Workflow Management based on State and Activity Charts," in Advances in Workflow Management Systems and Interoperability, Springer Verlag, 1998.
46. M.T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, 2nd edition, Prentice Hall: Englewood Cliffs, New Jersey, 1998.
47. M. Reichert and P. Dadam, "ADEPT$_{flex}$—Supporting dynamic changes of workflows without loosing control," Journal of Intel. Info. Systems, vol. 10, no. 2, pp. 93–129, 1998.
48. M. Rusinkiewicz, A. Cichocki, A. Sheth, and G. Thomas, "Bounding the effects of compensation under multi-level serializability," Dist. and Parallel Databases, vol. 4, no. 4, October 1996.
49. M. Rusinkiewicz and A.P. Sheth, "Transactional workflow management systems," in Proc. of Advances in Database and Information Systems, ADBIS'94, Moscow, May 1994.
50. M. Rusinkiewicz and A.P. Sheth, "Specification and execution of transactional workflows," Modern Database Systems: The Object Model, Interoperability and Beyond, W. Kim (Ed.), ACM Press: New York, NY, 1995, pp. 592–620.
51. F. Schwenkreis, "A formal approach to synchronize long-lived computations," in Proc. of the 5th Australasian Conf. in Information Systems, Melbourne, 1994.
52. A. Sheth, D. Georgakopoulos, S.M.M. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, and A. Wolf, Report from the NSF Workshop on Workflow and Process Automation in Information Systems. Accessible via: http://lsdis.cs.uga.edu/activities/
53. I. Stahl, Introduction to Simulation with GPSS, Prentice Hall, 1990.
54. J. Tang and S.-Y. Hwang, "Handling uncertainty in workflow applications," in Proc. of 5th Intl. Conf. on Info. and Knowledge Engineering, CIKM'96, Maryland, November, 1996.
55. J. Tang and J. Veijalainen, "Transaction-oriented workflow concepts in inter-organization environments," Intl. Conf. on Information and Knowledge Management, Baltimore, 1995.
56. H. Waechter and A. Reuter, The ConTract Model, in Database Transaction Models for Advanced Applications, Morgan Kaufmann Publishers: San Mateo, CA, 1992, chap. 7.
57. G. Weikum, "Principles and realization strategies of multilevel transaction management," ACM TODS, vol. 16, no. 1, 1991.
58. D. Wodtke and G. Weikum, "A formal foundation for distributed workflow management based on state charts," in Proc. of 6th Intl. Conf. on Database Theory, Greece, January 1997.
59. D. Worah and A. Sheth, "What do advanced transaction models have to offer for workflows?," Intl. Workshop on Adv. Transaction Models and Architectures, India, August 1996.

DISTRIBUTED AND PARALLEL DATABASES

An International Journal

INSTRUCTIONS FOR AUTHORS

Authors are encouraged to submit high quality, original work that has neither appeared in, nor is under consideration by, other journals.

PROCESS FOR SUBMISSION

1. Authors should submit five hard copies of their final manuscript to:

   Melissa Parsons
   DISTRIBUTED AND PARALLEL DATABASES

   Editorial Office
   Kluwer Academic Publishers — Tel.: 617-871-6600
   101 Philip Drive — FAX: 617-871-6528
   Norwell, MA 02061 — E-mail: kcollen@wkap.com

   For prompt attention, all correspondence can be directed to this address.
2. Authors are strongly encouraged to use Kluwer's LaTeX journal style file. Please see ELECTRONIC FORM section below.
3. Enclose with each manuscript, on a separate page, from three to five keywords.
4. Enclose originals for the illustrations, see "Style for Illustrations", for one copy of the manuscript. Photocopies of the figures may accompany the remaining copies of the manuscript. Alternatively, original illustrations may be submitted after the paper has been accepted.
5. Enclose a separate page giving the preferred address of the contact author for correspondence and return of proofs. Please include a telephone number, fax number and e-mail address, if available.
6. The refereeing is done by anonymous reviewers.

ELECTRONIC FORM

Upon acceptance of publication, *the preferred format of submission is the Kluwer LaTeX journal style file.* The style file may be accessed through a WWW site by means of the following commands:

WWW URL: http://www.wkap.nl

- Choose "Author Instructions/Stylefiles"
- Choose "Journals Listed Alphabetically"
- Choose appropriate journal listing
- Choose "Stylefiles" from list of options

Authors are encouraged to read the "readme" file.

Application of Information Technologies to the Maritime Industries, C. Guedes Soares, J. Brodda, (Eds.), Edições Salamandra, Lisbon, 1999, (ISBN: 972-689-157-4), pp. 33-51

MARIFLOW
A WORKFLOW MANAGEMENT SYSTEM FOR MARITIME INDUSTRY

Asuman Dogac, Catriel Beeri, Arif Tumer, Murat Ezbiderli, Nesime Tatbul, Cengiz Icdem, Guray Erus, Orhan Cetinkaya and Necip Hamali*

Software Research and Development Center, Faculty of Engineering
Middle East Technical University (METU), 06531 Ankara Turkiye
asuman@srdc.metu.edu.tr

* Institute of Computer Science, Hebrew University
91904 Jerusalem Israel
beeri@cs.huji.ac.il

Abstract

The aim of MARIFlow Project is to provide a prototype of an architecture for automating and monitoring the flow of control and data over the Internet among different organisations. This "electronic medium", capable of delivering value-added services to the participants, encompasses many different technological areas: from communication to security, databases, transaction support and agents. The project will make use of these technologies to produce a workflow management system. In particular, the goal of the project is to develop an adaptable workflow engine through which the activities of the different participants in the maritime industry can be harmonised, combined, and expanded through better tracking of functional dependencies and documents, improved data access and handling, and lower administrative overheads.

The MARIFlow system is based on *data-centric* approach, that is, execution of activities on a host machine is triggered by arrival of data, and generates further data that is sent to participants in the process, where upon arrival may trigger further activities. Nevertheless, the important characteristic of such processes is distribution, not just in terms of geography, but also in terms of ownership, responsibility and autonomy. This paper also addresses the important issues such as security of the documents as well as the tracking of data and documents, for monitoring purposes.

1 Introduction

Workflow systems, in general, provide for declarative means for specifying the control flow among activities and extensive research and development, both in the academia and in the industry have contributed to various aspects of the workflow system in making a mature technology - Alonso (1995), Georgakopoulos (1995), Miller (1997), Ming Shan (1998), Dayal (1998), Muth (1998), Sheth (1998), Vossen (1998), Alonso (1998), Cichocki and Rusinkiewicz (1998), and Dogac (1998). However in most of the systems, data flow is restricted to the parameters of the involved activities often disconnected from the description of the flow itself. In other words, the workflow management systems as used by industry today, use a process centric approach. Thus, they lack a mechanism with which it is possible to define the source of data, control its flow over the net, and identify and possibly invoke the activities that make use of it. The workflow engine to be developed as part of this project will address these crucial issues with a special emphasis on the ease of use, maintenance, and customization to the needs of maritime industry.

In maritime industry, materials used in shipbuilding or repairs need to be certified by a classification society. In current practice, the material is checked while it is at the production plant and the "quality data", related to this material, is delivered to the classification society. If the quality data fulfills the requirements, a paper certificate is issued and delivered to the production plant as well as to the customer. Once issued the certificate follows the material to the main shipyard or to one of the subcontractors from where it is eventually added to the ship's documentation file. The certificate is checked at every production stage as well as at ship's handover and at each survey during ship's life cycle. In an industry involving the flow of large amount of paper documents among different organizations, this is a slow, expensive, tedious, error-prone and very limiting process, which in some cases can hinder the ability to improve the service quality. In this work we describe an architecture that provides for automating and monitoring the flow of control and data over the Internet among different organizations and companies with special attention to maritime industry.

In the MARIFlow system, the higher order process is defined through a graphical user interface, which is then mapped to a textual language called FlowDL. FlowDL allows indicating the source of the documents, their control flow and the activities that make use of these documents. A process definition in FlowDL is executed through co-operating agents, called MARCAs (MARIFlow Co-operating Agents), that are automatically initialized at each site that the process executes. The initiation of agents and monitoring facilities are managed through Java programs which can be used by authorized users through Web. The main Web page through which the tools can be started is shown in Figure 1.

MARCAs are responsible for handling the activities at their sites, routing the documents in electronic form, according to the process description among other MARCAs, keeping track of process information by logging activities, and providing for the security and authentication of documents during communication.

Figure 1. MARIFlow MARCA Tools Web Page

The overall system should be immune to failures, and the processing power should be distributed such that it will not generate a serious bottleneck built around a single machine, which is accessed extensively. The participants of the workflow system should decide and behave on their own rather than being invoked or commanded by other programs in a centralized fashion. These requirements necessitate an agent-based architecture for the workflow system, where independent entities capable of completing complex assignments without intervention are used rather than tools that must be manipulated by a user.

In the MARIFlow system, the responsibilities of MARCAs in order to support the requirements given above are defined as follows:

- A MARCA receives messages through a persistent queue and evaluates them to decide what action to take. The persistent queue is necessary so that the agent does not lose its state after a program crash, site failure etc.
- It persistently stores the documents it receives. If the organization that the MARCA resides on has a firewall mechanism, it is also MARCA's responsibility to pass the documents to the in-house system, get the

resulting documents from the system and forward them to the related agents as specified in the process definition.

- Process related information should be stored persistently for further monitoring purposes. Therefore it is MARCA's responsibility to direct related information to a database through its JDBC interface.
- A process definition is compiled at a host and through a special MARCA the information is distributed over the system to other MARCAs necessary for the given workflow definition at initialization phase. That special MARCA is also responsible for data warehousing for monitoring purposes.

This paper is organized as follows: In Section 2, the general architecture of the system is described. This section introduces the MARIFlow Cooperating Agents (MARCAs), the FlowDL workflow definition language, monitoring of the workflow processes along with an example definition for maritime industry to illustrate to details of the architecture. Section 3 describes how security and authentication of documents are handled in MARIFlow. The availability and scalability issues are discussed in Section 4. In Section 5, the work that remains to be done is described namely the persistency of the messages to recover from failures and compensation of activities. Finally, Section 6 concludes the paper.

2 The Architecture of the System

2.1 An Overview of the Architecture

Figure 2 gives an overview of the general architecture of the MARIFlow system. Each organization may have in-house applications inside a firewall protected from unauthorized access. MARCAs exist on a host machine outside the firewall relative to the site network. The MARIFlow agent informs in-house applications when necessary through internal process initiator and is responsible for sending and receiving documents and process related information through the firewall mechanism.

In MARIFlow an inter-enterprise workflow is defined graphically where the workflow designer specifies domains, tasks and process information which are used in building the process definition. The graphical representation of the process definition is mapped to FlowDL, the language used in MARIFlow, and from that definition the specifications for each MARCA is generated. The compilation of the process definition is done on the coordinating MARCA, which is installed in one of the sites. The behavioral structure, obtained from the process definition, for each MARCA is then transmitted to corresponding agent and MARCAs become available for distributed workflow management. It should be noted that this transmission

along with the communication at instance level should be realized through persistent queues in order to survive through system crashes and other problems.

2.2 MARIFlow Cooperating Agents: MARCAs

A workflow instance in MARIFlow is executed by co-operating agents called MARCAs. There is exactly one MARCA at each site participating to the workflow execution and it handles all the activities running at its site. At initialization, once in their life time, the sites download the generic MARCA template from a given URL. At compilation time the guards of activities within the responsibility of a MARCA are determined according to the process definition. Guards are special mechanisms based on intertask dependencies - Attie (1993), Singh (1996). They are logical expressions for significant events of activities of a MARCA like "start" or "terminate". MARCAs evaluate these guards with the messages that they receive to decide on their actions. In other words, guards inform the MARCA when to execute a certain activity. A detailed formal description of obtaining guard expressions from a given workflow specification is given in Dogac (1998). All of the information about guard structures is obtained from the process definition at the initialization phase of the MARCA.

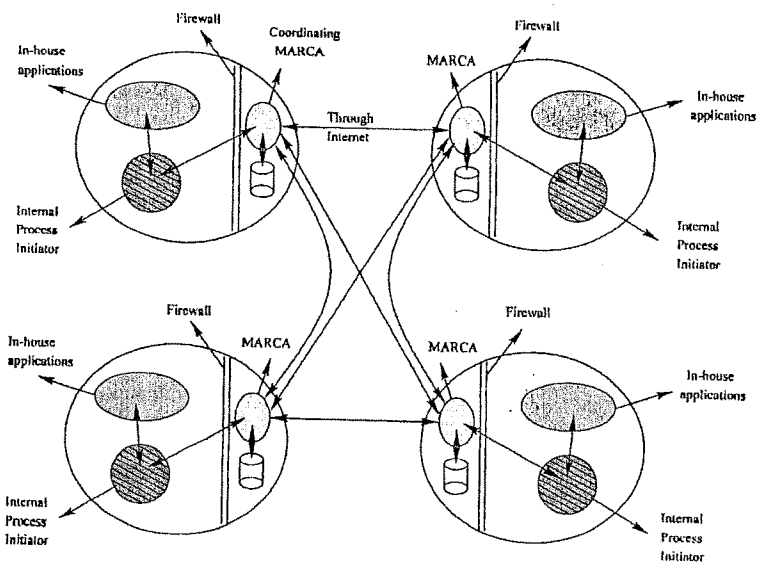


Figure 2. An Overview of the Architecture

There is a special MARCA in the workflow system called the coordinating MARCA. A workflow definition is realized through the coordinating

MARCA and compiled only once. During this compilation, guard structure for each MARCA participating the system along with the interactions with the other MARCAs are obtained and the MARCAs are initialized with this information by the coordinating MARCA. A single MARCA residing on a host is capable of handling multiple workflow definitions and multiple instances of a given workflow at a certain time. All messages are differentiated by unique workflow id obtained from the definition and instance ids that are automatically assigned by the system for each instance generated.

MARIFlow agents communicate with each other through TCP/IP over the Internet. Network concurrent accesses to a single port is handled by Java's Net Package by assigning dynamic ports for each request. Simple message buffering and queuing are also provided by this package. These queuing facilities have been extended to persistent queue implementation and transactional agent communication for safe and consistent transmission.

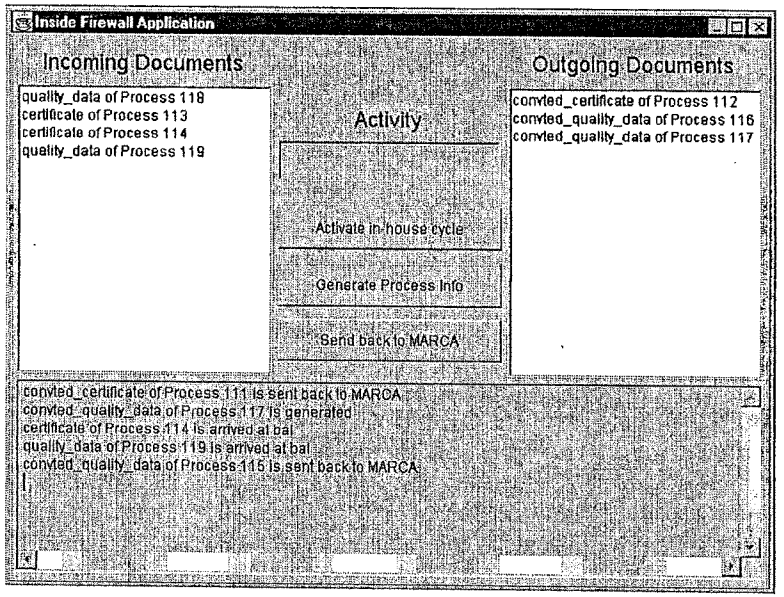


Figure 3. Graphical User Interface for Process Control inside the Firewall

MARCAs communicate with each other through an Agent Communication Language, which is specific to MARIFlow agents, hence they do not communicate with agents in the outside world. This is because communicating with outside world agents is not necessary within the scope of the work. During a session, each message is preceded by an activity identifier along with workflow and instance ids. This information is

necessary to identify how a received document or message should be managed according to the activity identifier.

A MARCA sends a document and process related information inside the firewall through an e-mail. Since the content of the documents may be binary, they are encoded prior to the attachment by traditional mime base64 encoding used in e-mail messages.

A program inside the firewall processes the incoming mail and extracts the documents, process related information, workflow and instance ids, the activity that will use this document inside the firewall and displays this information through a GUI as shown in Figure 3. When a document arrives through an e-mail it is shown in the *Incoming Documents* list of the interface. When a document is highlighted (i.e. selected), the activity that will use this document appears in the *Activity* box. The *Activate in-house Cycle* button may be used to start the given activity provided that API of the activity is available. *Send back to MARCA* button is used to transfer the document selected in the *Outgoing Documents* list to the MARCA outside if the related information form of the document is filled via the interface displayed when *Generate Process Info* button is used.

We have chosen to send the response back to MARCA through e-mail mechanism to be system independent as much as possible, since some firewall architectures disallow packet transmission in both directions. The reader program for the mailbox is based upon POP3 and receives the messages through the POP3 server port. This choice is also an effort to be system independent in MARIFlow since POP3 is independent of the structure of the mailbox in different operating system implementations.

2.3 *Workflow Definition Language: FlowDL*

In MARIFlow system, a workflow process definition is given in FlowDL as a collection of blocks, tasks and other sub-processes as well as some explicit declarations and commands to specify Internet domain addresses, sources of documents, process specific information to be used for monitoring the document flow, and activities for further processing on the documents. The term *activity* is used to refer to a block, a task, or a (sub)process.

FlowDL contains several kinds of blocks, which are used to define different kinds of flow types in the process definition: the activities that run in parallel, in serial or under conditions etc. These blocks along with the declarations done in the process definition code define the whole workflow system. The blocks types encapsulate the workflow primitives defined in Hollingsworth (1996), which are sequential, AND-split, AND-join, OR-split, OR-join and repeatable task.

In MARIFlow, an inter enterprise workflow is defined graphically by using the tool as shown in Figure 4. This tool allows the workflow designer to

make the declarations, specify domains, tasks and process information. Afterwards, different types of blocks can be generated and added to the workflow by using the declarations in order to preserve consistency. The workflow definition is mapped to the textual FlowDL language. When this definition is parsed, the guards of each MARCA participating the system are generated and the agents are initialized with this information. The guards provide for the behavioral definitions of the MARCAs.

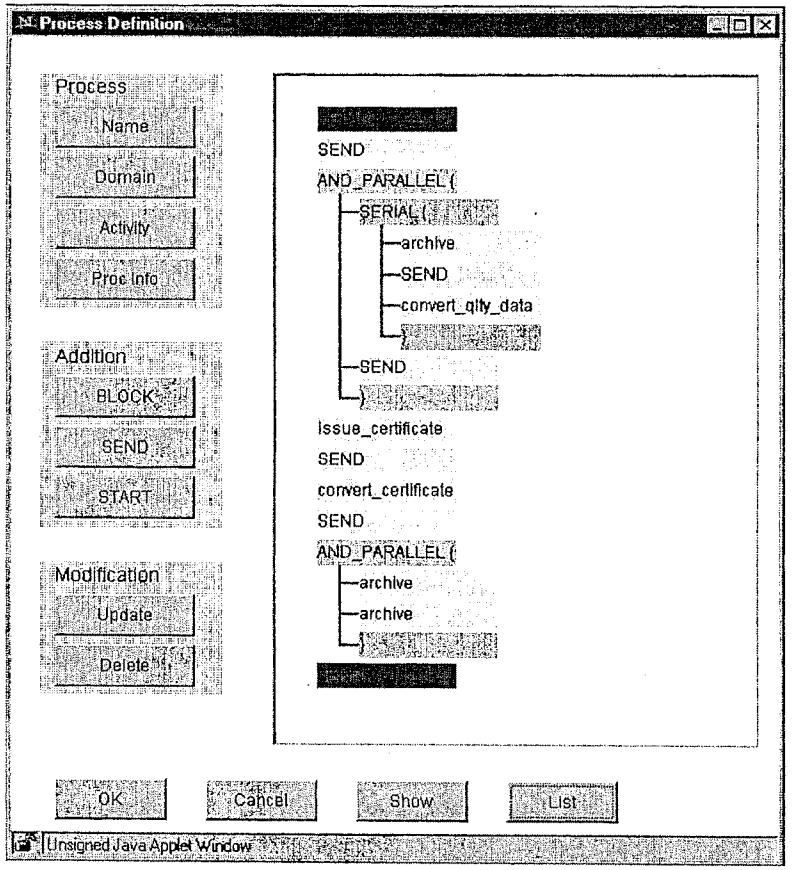


Figure 4. Graphical User Interface for Process Definition in MARIFlow

The advantages brought by block structured FlowDL language can be summarized as follows:

- The current workflow specification languages are unstructured and/or rule based, as noted in Sheth (1996). Unstructured languages make debugging/testing of a complex workflow difficult and rule based ones become inefficient when they are used for specification of large and complex workflow processes due to the large number of rules and overhead associated with rule management. FlowDL avoids these disadvantages through its block-structured nature.
- A block structured language confines the inter-task dependencies to a well-formed structure that in turn proves extremely helpful in generating the guards of activities for distributed scheduling of a workflow.
- Blocks not only clearly define the data and control dependencies among tasks but also present a well-defined recovery semantics through compensation for a failed or aborted block.

In addition to activities, there are also assignment statements in FlowDL which access and update workflow relevant data. In this way the workflow designer has the ability to assign values to variables. Note that these variables are used in conditional and loop blocks.

2.4 Monitoring of Workflow Processes

Each MARCA stores all the messages it receives in a persistent log so that after a crash or a site failure the MARCA can be brought back to a consistent state by using the information in the log. The MARCAs also store additional process related information specified by the workflow designer through FlowDL inside the database system. Since Java is used in coding the MARIFlow system, a Java native JDBC interface is used for database connectivity. Consequently any database with a JDBC interface can be used by the MARCA.

Each MARCA sends a copy of the information it stores to the coordinating MARCA. Thus coordinating MARCA constitutes a data warehouse site for monitoring information. This site is available to any authorized user on the Internet through a Web interface and may further be replicated for availability purposes.

The authorized user can track the flow of a process instance through a graphical user interface as shown in Figure 5 by giving the process instance identifier and the workflow definition it belongs to. The Web interface reads the process definition of the selected workflow from the database, and produces a graphical representation from that information, with different colors for each block type and with lines connecting the blocks showing the flow of data and messages. Also the interface reads the instance information from the coordinating MARCA's database and reflects it on the graph so that the user can see the instant state of the process instance on the screen, along with the finished, on going and yet to be started activities.

The information kept in the database system of the coordinating MARCA can be queried through Java Applets directly from the Web.

The advantages of this monitoring architecture are as follows:

- It provides for high availability since the data is stored both in the MARCAs locally in a distributed manner and also in a data warehouse in a centralized manner.
- Authorized user can query the data warehouse from anywhere on the Internet by using a simple Web browser.
- Response to monitoring queries will be fast since data is obtained from a single store rather than performing distributed query processing.

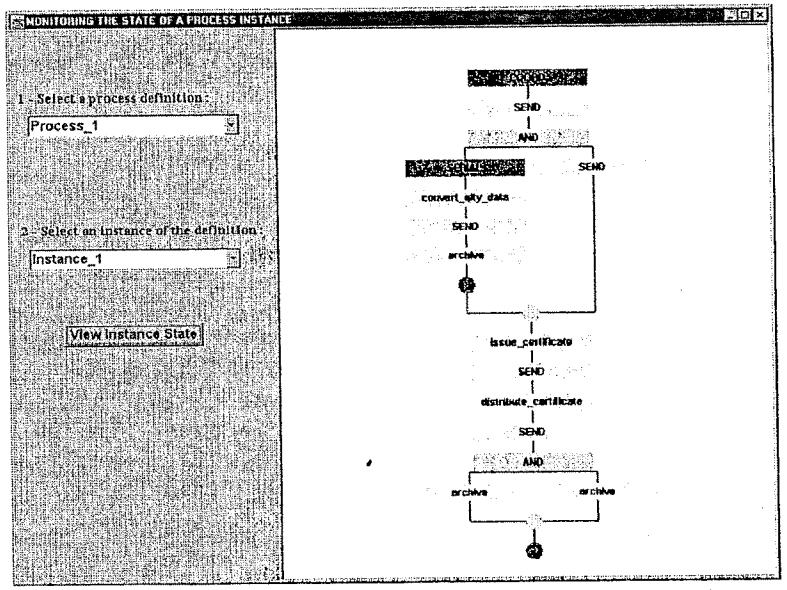


Figure 5. Graphical Monitoring Interface of the MARIFlow System

The databases of each MARCA can also be queried locally by authorized users.

2.5 *The* Certification *Process Definition*

The following is an example workflow defined in FlowDL reflecting a business scenario for a certification process in maritime industry:

```
PROCESS Certification ();

ACTIVITY archive (IN document arch_document);
ACTIVITY convert_qlty_data (IN document quality_data
        OUT document convted_q_d);
ACTIVITY convert_certificate(IN document certificate
        OUT document convted_cert);
ACTIVITY issue_certificate (IN document conv_quality_data
        IN document product_spec OUT document certificate);
ACTIVITY delete_from_archive();
ACTIVITY cancel_converted_data();
ACTIVITY cancel_certificate();

DOMAIN_DEFINITION {
        salzgitter_ag.de        szag;
        germanlloyd.org         gl;
        balance_bremen.de       bal;
        isisanisi.com           isisan;
}

struct process_info {
        string order_no;
        string material_no;
        string customer_name;
        string supplier_name;
        string class_society_name;
        string certificate_number;
}

DEFINE_PROCESS MARIFlow ()
{
        szag SENDS (quality_data) TO bal GENERATES (order_no
                material_no customer_name supplier_name
                class_society_name);
        AND_PARALLEL {
                SERIAL {
                        START convert_qlty_data (IN quality_data OUT
                                convted_q_d) AT bal COMPENSATED BY
                                cancel_converted_data();
                        bal SENDS (convted_q_d) TO gl AND isisan;
                        START archive (IN convted_q_d) AT isisan NON_VITAL
                                COMPENSATED BY delete_from_archive();
                }
                isisan SENDS  (prod_spec) TO gl;
        }
        START issue_certificate (IN convted_q_d IN prod_spec OUT
                certificate) AT gl GENERATES (certificate_number)
                COMPENSATED BY cancel_certificate();
        gl SENDS (certificate) TO bal;
        START convert_certificate (IN certificate OUT convted_cert)
                AT bal;
```

```
    bal SENDS (convted_cert) TO szag AND isisan;
    AND_PARALLEL {
        START archive (IN convted_cert) AT szag NON_VITAL
            COMPENSATED BY delete_from_archive();
        START archive (IN convted_cert) AT isisan NON_VITAL
            COMPENSATED BY delete_from_archive();
    }
}
```

Example 1. An Example Workflow Definition for the Maritime Industry

The process starts when the steel company (*szag*) sends the "quality data" to a service company (*bal*) to be transformed into EDIFACT (Electronic Data Interchange for Administration, Commerce and Transport) standard. Then within the scope of a block the following activities run in parallel:

- At *bal*, "quality_data" is converted into EDIFACT standard by invoking "convert_qlty_data" activity. The document produced, "convted_q_d", is sent to *gl* and *isisan*. At *isisan*, the arrived document is archived by invoking *archive* activity. Note that the block that comprises these activities is a SERIAL block where the items in the block are executed sequentially.
- The steel user (*isisan*) sends the product specification document to the classification society (*gl*).

Once *gl* receives the converted quality data, its system is notified to start the "issue_certificate" process. When the issued certificate, "certificate", is generated and delivered to the MARCA outside the firewall, it is transferred to *bal* again for conversion to EDIFACT standard and this document is sent to *szag* and *isisan*. Afterwards "archive" activities at *isisan* and *szag* are started and executed in parallel for converted certificate, "convted_cert". This completes the cycle of one instance in *Certification* process definition.

The guard structures for each MARCA is generated when the process definition is parsed and process tree is created as in Figure 6. The guards evaluate to true when the necessary messages and/or documents arrive at MARCAs either from the network or from the inside firewall application. Hence the necessary action is taken such as an activity is started or the execution through the blocks continues. In the running example, when "prod_spec" arrives at *gl* the guard still evaluates to false since for the guard to evaluate to true the arrival of document "convted_q_d" is also necessary. Therefore, only when both of the documents arrive at *gl* the guard evaluates to true and the necessary course of action is taken, that is, the "issue_certificate" task is invoked, to create the "certificate".

For monitoring purposes, each MARCA stores the following process related information persistently in a database system: order_no, material_no,

customer_name, supplier_name, class_society_name and certificate_number. Note that the activity responsible for providing a specific piece of information, such as order_no, or certificate_number is obtained through the GENERATES statement of the activities. For example the activity "START issue_certificate (IN convted_q_d IN prod_spec OUT certificate) AT gl GENERATES (certificate_number);" declares that certificate_number will be produced by this activity.

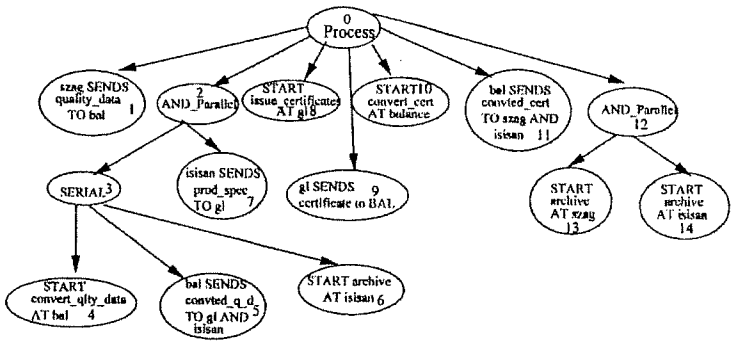


Figure 6. Process Tree of the Example Process Definition

Some example queries that can be readily answered by the database system include: Status of a certificate given its number, status of all certificates for a given steel order, number of certificates issued by the classification society etc. These queries can be executed through the User Interface specialized for this process definition, using any World Wide Web Browser capable of running Java Applets. Figure 7 depicts a customized GUI available for the MARIFlow project.

3 Authentication and Authorisation

Conducting business over a public network, like the Internet, requires mechanisms for authentication of messages, for authorization to access data and execute operations, and for privacy and security in general. We discuss briefly some of the requirements in our application domain, and how they relate to our software architecture.

Clearly, activities that are performed inside a company's firewall are irrelevant to our scenario; a company may use whatever mechanisms it chooses. Specific requirements arise with respect to company-to-company operation, to distributed intra-company operation, and to the overall operation of the MARCA network. We consider each in turn.

An example of a company-to-company operation is sending a quality document from a manufacturer to a classification company. In our scenario,

this involves three companies, since a service company translates the document before being sent to the classification society. When a set of companies are involved in a well-defined business procedure, they naturally want the details to be protected from potential competitors. The obvious solution is to encrypt documents inside the firewall, using a scheme that is common to all involved parties. Such encryption also provides authentication of the sender of a message that can easily be verified by the receiver. Finally, an unauthorized receiver will not be able to read a message.

A common situation in our scenario is that representatives of the classification society are present in manufacturing sites, as part of the certification process. These representatives need access to documents of materials and parts, from their society's repository. Although this is an intra-company operation, it is similar in nature to the previous case. An intra-company scheme can be used to encrypt documents before they are passed outside the firewall. An encryption of the request for a document from a field agent can provide authentication, as well as privacy, since the message contents, namely the actual request, is then also not easily available to external entities.

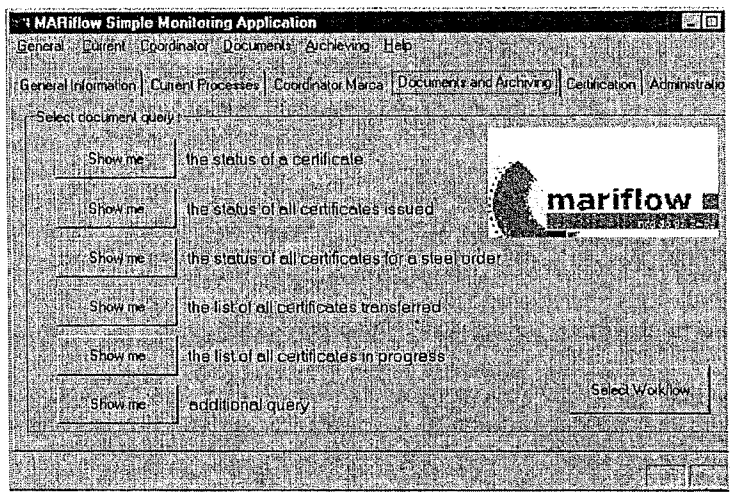


Figure 7. A monitoring Interface specific to the given Example Definition

In both situations above, as the MARCAs need to know the sources and destinations of messages, and the process to which they belong to, such details will have to be provided additionally to the encrypted message. As the MARCA system is outside the firewall, it cannot participate in the internal encryption scheme. To further increase the level of protection and of authentication, i.e., the certainly regarding the sources of received messages; the MARCAs use an independent encryption scheme. While this provides a

reasonable degree of protection against casual listeners, clearly it does not provide the same level as the internal schemes, simply because the MARCAs are outside the firewalls, hence their code is open for analysis. Nevertheless, we believe that the solution is satisfactory, given the additional level of protection and authentication provided by the intra company systems. The alternative, namely the development of a trusted authority to manage protection and authentication schemes for all involved companies, requires both further research, such as development of virtual firewall for clusters of companies, as well as a rather drastic change in business procedures.

4 Availability and Scalability

When a site goes down, restarting the MARCA is under the responsibility of the Operating System's start up control. The site's start up control is to analyze the persistent logs of the MARCA and start a new instance using these stable logs created before the site crash. However, there is need for a further mechanism to prevent any Operating System related problem.

In MARIFlow, for each MARCA installed there is a background process at that site, called the "rescue process". The rescue process is responsible for monitoring the lifetime of the agent and checks the MARCA at specific time intervals through a predetermined socket. A thread of the MARCA listens to this socket and responds to the signals. If the MARCA does not respond to this process for a given period of time, the process starts sending signals more frequently. If the MARCA still does not respond, after sending a bunch of signals the process assumes that the MARCA is not functional. The two possibilities in this case are the MARCA could be blocked or it could be dead. When the rescue process is unable to find the OS process that belongs to this MARCA (i.e., it is dead), it instantiates a new MARCA by the help of the persistent logs related with the state of the agent.

Otherwise if the MARCA is blocked, it is necessary to kill the old instance prior to installation of a new instance. Since the logs are persistent it is possible to recover the state of the MARCA killed and hence the site does not suffer from any inconsistencies.

For the described mechanism to work correctly it is necessary to make sure that rescue process stays alive. Therefore, just as the rescue process checks to see that the MARCA stays alive, the MARCA also checks to ensure that the rescue process stays alive by signaling the rescue process at predefined time intervals. It is MARCA who re-instantiates the rescue process when it dies.

5 Future Work

The current prototype of the system does not yet include the mechanisms for the following issues:

5.1 Compensation

Compensation activities are used to logically undo the effects of the activities with which they are associated (Dayal, 1991). They are used when a failure occurs in the system and the system should be taken to a stable state before the failure. On the other hand, workflow processes are long-running activities consisting of many nested sub-activities. In an activity hierarchy, the failure of a sub-activity may cause its parent or higher level ancestors to abort. However, it is not acceptable to roll all the finished activities back in case of failures. A hierarchical approach to failure handling which allows for partially rolling back to the nearest point in process history tree where it is possible to restart execution is required. Moreover, failures should be handled in a timely and efficient fashion. Chen and Dayal (1996) provide such a mechanism that can be adapted into the MARIFlow system. It provides a failure handling mechanism consisting of two phases. When a sub-activity *T* fails, it is necessary to determine the impact of that failure on the ancestors of *T* by finding out the highest level ancestor that should be aborted. The root of the activity sub-tree to be logically undone upon *T*'s failure is called the *Logical-Undo Root(LUR)* of *T*. Every activity in an activity hierarchy has a corresponding *LUR*, which may be one of the following:

- the closest non-vital ancestor since its failure can be ignored by its parent,
- the closest ancestor with a contingency activity (children of a contingency block),
- the closest ancestor without a parent, which may be the top-level process or a compensation activity.

Bottom-up searching for *LUR* in the process tree constitutes the first phase of the approach. After the *LUR* is found, the effects of the activities in the sub-tree with *LUR* as the root are removed in a top-down fashion. Applying the undo operation top-down provides a timely reaction to a failure by halting the activities in scope of *LUR* promptly. In this second phase, starting from the *LUR*, the finished activities are compensated (if they have compensation activities) taking the semantics of the blocks that enclose them into consideration. The approach also allows compensations to be made as high level as possible since compensating a high-level activity is more general than compensating a lower-level activity. After this two-phased

algorithm is applied to the activity hierarchy, the execution restarts and rolls forward from the next activity after *LUR*.

5.2 Exception Handling

Hagen and Alonso (1998) propose a flexible approach to exception handling. In this approach, the business logic is separated from exception handling logic, which makes it easier to keep track of each. Exception handling code is separated from the normal code, which also provides reusability of components in addition to simplicity.

In Hagen and Alonso (1998), exceptions are treated as separate objects. Each has a name and parameters. Each exception type should be registered with the system giving its name and interface. Also, each one has a category that indicates the behavior of the handler with which it is associated. An exception handler is a special process that is started when an exception has been signaled. At the beginning, workflow components (activities), exceptions and handlers each handling an associated exception are defined in the system. Then, a workflow process is composed using these components. This component architecture approach provides both reusability and flexibility where different components can be used in different combinations in different workflow definitions. However, the approach does not have a solution for handling unpredictable events that may occur at execution time. All the exception cases and their handlers are given at the beginning.

6 Conclusions

In this paper we described an architecture for a workflow system for document flow over the Internet realized through cooperating agents. The architecture provides for the declarative specification and automatic generation of the application rather than producing large amounts of application specific code by the help of block structured nature of the workflow definition language.

The system attempted to bring solutions to user interaction through the Java based Web interfaces, reliable and automated document and data transfers through a distributed agent based architecture and it is more resistant to localized failures than that of centralized systems. The block structured nature of the language avoids unreachable states in the workflow execution and also the deadlocks (Alonso, 1995). The system also proposes an adaptable design and implementation so that the engine will run in different platforms ranging from a small set of personal computers to high end workstations and mainframes by the universal programming language Java.

The architecture is general enough to be applied to any domain however the example application is provided for maritime industry and therefore some of

the user interfaces are customized accordingly. Nevertheless the core of the system, workflow specification, communication details and database activities are suitable for any kind of domain definition thanks to the layered architecture.

The first prototype of the system including the details that are worked around in this paper is available. The issues like compensation and exception handling are yet to be tackled.

References

G. ALONSO, R. GUNTHOR, M. KAMATH, D. AGRAWAL, A. El ABBADI, C. MOHAN, (1995), "Exotica/FDMC: A Workflow Management System for Mobile and Disconnected Clients", *Parallel and Distributed Databases*, The Netherlands.

G. ALONSO, C. HAGEN, H. SCHEK, M. TRESCH, (1998), "Towards a Platform for Distributed Application Development", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 195-221.

P. C. ATTIE, M. P. SINGH, A. SHETH, M. RUSINKIEWICZ, (1993), "Specifying and Enforcing Intertask Dependencies", *Proceedings of the 19th VLBD*, Dublin, Ireland

Q. CHEN, U. DAYAL, (1996), "A Transactional Nested Process Management System", *in Proceedings of 12th International Conference on Data Engineering*, New Orleans, LA.

A. CICHOCKI, M. RUSINKIEWICZ, (1998), "Migrating Workflows", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 339-355.

U. DAYAL, M. HSU, R. LADIN, (1991), "A Transactional Model for Long-Running Activities", *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona

U. DAYAL, Q. CHEN, TAK W. YAN, (1998), "Workflow Technologies Meet the Internet", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 423-438.

A. DOGAC, E. GOKKOCA, S. ARPINAR, P. KOKSAL, I. CINGIL, B. ARPINAR, N. TATBUL, P. KARAGOZ, U. HALICI, M. ALTINEL, (1998), "Design and Implementation of a Distributed Workflow Management System: METUFlow", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A Sheth (Eds), Springer-Verlag pp. 61-91.

D. GEORGAKOPOULOS, M. HORNICK, A. SHETH, (1995), "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", *Distributed and Parallel Databases*, Ahmed K. Elmagarmid (Ed-in-chief), Volume 3, Number 2, pp. 119-153.

C. HAGEN, G. ALONSO, (1998), "Flexible Exception Handling in the OPERA Process Support System", *18th International Conference on Distributed Computing Systems (ICDCS 98)*, Amsterdam, The Netherlands.

D. HOLLINGSWORTH, (1996), "The Workflow Reference Model", *Technical Report TC00-1003, Workflow Management Coalition*, Accessible via: http://www.aiai.ed.ac.uk/WfMC/

J. MILLER, D. PALANISWAMI, A. SHETH, K. KOCHUT, H. SINGH, (1997), "WebWork: METEOR$_2$'s Web-based Workflow Management System", *Journal of Intelligent Information Systems*, The Netherlands

P. MUTH, D. WODTKE, J. WEISSENFELS, G. WEIKUM, A. DITTRICH, (1998), "Enterprise-Wide Workflow Management Based on State and Activity Charts", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 281-303.

MING-CHIEN SHAN, J. DAVIS, W. DU, Y. HUANG, (1998), "HP Workflow Research: Past, Present, and Future", *in Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 92-106.

A. SHETH, D. GEORGAKOPOULOS, S. JOOSTEN, M. RUSINKIEWICZ, W. SCACCHI, J. WILEDEN, A. WOLF, (1996), "Report from the NSF Workshop on Workflow and Process Automation in Information Systems", *SIGMOD Record*, 25(4):55-67

A. SHETH, K. KOCHUT, (1998), "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems", in *Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 35-60.

M. SINGH, (1996), "Synthesizing Distributed Constrained Events from Transactional Workflow Specifications", *in Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, New Orleans

G. VOSSEN, M. WESKE, (1998), "The WASA Approach to Workflow Management for Scientific Applications", *in Workflow Management Systems and Interoperability*, NATO ASI Series, A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds), Springer-Verlag pp. 145-164.

The MARIFlow Workflow Management System *

A. Dogac, M. Ezbiderli, Y. Tambag, C. Icdem, A. Tumer, N. Tatbul, N. Hamali
Software R&D Center, METU, Ankara, Turkiye, asuman@srdc.metu.edu.tr
C. Beeri
Hebrew University, Jerusalem, Israel

MARIFlow System [1] provides for automating and ıonitoring the flow of control and data over the Internet ıong different organizations, thereby creating a platform ɔessary to describe higher order processes involving sev- ıl organizations and companies.

The architecture is general enough to be applied to any siness practice where data flow among different indus- es and cooperations and the invocation of activities follow ɔattern that can be described through a process definition. ıe example application provided within the scope of this oject is on maritime industry

A MARIFlow process is executed through cooperating ɛents, called MARCAs (MARIFlow Cooperating Agents) ıat are automatically initialized at each site that the process ɛecutes. MARCAs handle the activities at their site, pro- ide for coordination with other MARCAs in the system by ɔuting the documents in electronic form according to the rocess description, keeping track of process information, nd providing for the security and authentication of docu- ıents as well as comprehensive monitoring facilities.

More specifically, the functionality provided by the sys- ɜm is as follows:

- A declarative means to specify the control of document flow over the Internet where it is possible to define the source of data, its control flow and the activities that make use of this data.
- A fully distributed execution architecture achieved through cooperating agents over the Internet. The agents know about other agents that they need to communicate with and preserve their state during communication. They also manage local information for monitoring purposes and for recovering from failures.
- Communicating with inside firewall applications. A MARCA can activate in-house activities automatically. However it should be noted that most organizations may be reluctant to grant access inside the corporate firewall. In such cases, the MARCA passes the documents to an in-house system by properly acknowledging the in-house system on further processing that may be necessary on the documents. A MARCA is also responsible for getting the documents from the in-house system and forwarding them to the related agents as specified in the process definition.
- There is a coordinating MARCA in the system through which it is possible to define processes graphically from a Web interface. The coordinating MARCA is also responsible for initializing the MARCAs at each site for a new process definition and acting as a facilitator among MARCAs in the sense that for a new workflow definition it decides which new MARCAs are necessary. Note that only one MARCA exists at each site and handles all the activities of all workflow definitions related with that site. Therefore a new MARCA is generated only for a site participating to a workflow definition for the first time. Coordinating MARCA also acts as a data warehouse for monitoring purposes.
- Authentication and authorization of documents and the process related information.
- A monitoring mechanism for keeping track of the documents and for providing detailed account of the current status of a process instance within the system.
- Ability to recover the system from various type of failures.

* This work is being supported by the European Commission Project Number: INCO-DC 97-2496 MARIFlow, by the Middle East Technical University Project Number: AFP-97-07-02-08, and by the Scientific and Technical Research Council of Turkey, Project Number:197E038

References

[1] A. Dogac, M. Ezbiderli, N. Tatbul, C. Icdem, G. Erus, O. Cetinkaya, A. Tumer, C. Beeri, "A Workflow System through Cooperating Agents for Document Flow over the Internet", Submitted for publication

0-7695-0506-6/00 $10.00 © 2000 IEEE

A Workflow System through Cooperating Agents for Control and Document Flow over the Internet *

A. Dogac[1], Y. Tambag[1], A. Tumer[1], M. Ezbiderli[1], N. Tatbul[1], N. Hamali[1], C. Icdem[1] and C. Beeri[2]

[1] Software Research and Development Center
Middle East Technical University (METU), 06531 Ankara Turkiye
asuman@srdc.metu.edu.tr
[2] Hebrew University, Jerusalem, Israel
beeri@cs.huji.ac.il

Abstract. In this paper we describe an architecture that provides for automating and monitoring the flow of control and document over the Internet among different organizations, thereby creating a platform necessary to describe higher order processes involving several organizations and companies. The higher order process is designed through a graphical user interface and is executed through cooperating agents that are automatically initialized at each site that the process executes. Agents handle the activities at their site, provide for coordination with other agents in the system by routing the documents in electronic form according to the process description. The system is capable of activating external applications (which may be inside the company firewall) when necessary, keeping track of process information, and providing for the security and authentication of documents as well as comprehensive monitoring facilities. The architecture is general enough to be applied to any business practice where data flow and invocation of activities among different industries and cooperations follow a pattern that can be described through a process definition, however since the project is on maritime industry, some of the graphical user interfaces are customized accordingly. The system is fully operational for industrial use.

1 Introduction

In the MARIFlow system described in this paper, the higher order process is defined through a graphical user interface which is then mapped to a textual language called FlowDL. FlowDL is a block structured language encapsulating the six primitives defined by the Workflow Management Coalition through its blocks with which it is possible to describe flows and hence construct a workflow specification [3]. FlowDL allows to indicate the source of the documents, their control flow and the activities that make use of these documents.

A MARIFlow process is executed through cooperating agents, called MARCAs (MARIFlow Cooperating Agents) that are automatically initialized at each site that the process executes. MARCAs handle the activities at their site, provide for coordination with other MARCAs in the system by routing the documents in electronic form according to the process description, keeping track of process information, and providing for the security and authentication of documents as well as comprehensive monitoring facilities.

The responsibilities of the agents (MARCAs) in our architecture are as follows:

- A MARCA receives messages through a persistent queue and evaluates them to decide what specific action to take.
- It persistently stores the documents it receives. It should be noted that the organizations may be reluctant to grant access inside the corporate firewall. In such cases when the need arises, the MARCA passes these documents to an in-house system by properly acknowledging the in-house system on further processing that may be necessary on the documents. The MARCA is also responsible for getting the documents from the in-house system and forwarding them to the related agents as specified in the process definition.
- Process related information also needs to be stored persistently for monitoring purposes. In our system MARCAs store the information related with monitoring to any JDBC compliant database to be accessed through a JDBC interface.
- There is a single MARCA at each site that handles all the activities of all workflow definitions and their instances related with that site. Therefore a new MARCA is generated only for a site participating to the system for the first time.

If we summarize, the functionality provided by the system developed is as follows:

- A declarative means to specify the control and document flow over the Internet where it is possible to define the source of data, its control flow and the activities that make use of this data.
- Invoking external applications (which may be inside the company domain) when necessary.
- Authentication and security of documents and the process related information.
- A monitoring mechanism for keeping track of the documents and/or for providing detailed account of the current status of a process instance within the system.
- Measures for failure recovery and exception handling.

In the following the system is described very briefly due to space limitations. Interested reader is referred to [2] where full design and implementation details are presented.

* This work is being supported by the European Commission Project Number: INCO-DC 97-2496 MARIFlow, by the Middle East Technical University Project Number: AFP-97-07-02-08, and by the Scientific and Technical Research Council of Turkey, Project Number:197E038

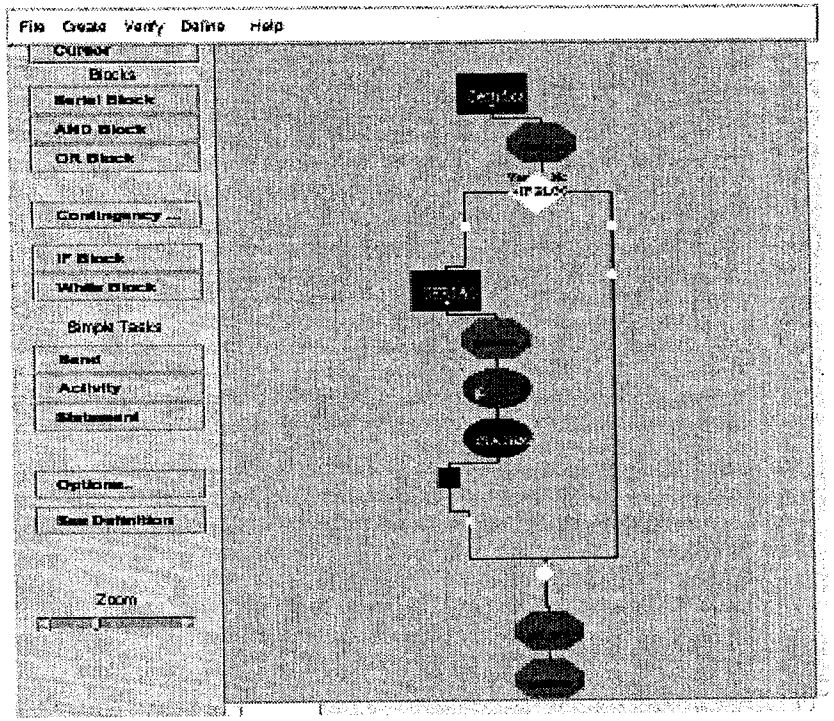


Fig. 1. The Graphical User Interface for Process Definition in MARIFlow

2 The Architecture of the System

In MARIFlow system each organization may have in-house applications inside a firewall protected from unauthorized accesses. MARCAs exist outside the firewall and inform in house applications when necessary through a User MARCA Interfacing Application (UMIA).

An inter enterprise workflow is defined graphically as shown in Figure 1. This tool allows the workflow designer to specify domains, tasks and process information which are then used in building the process definition graphically. This definition is mapped to the textual FlowDL language. The information on sites at which a MARCA should be installed are obtained from the domain definition in the process specification. These sites download a generic MARCA from a given URL. At compilation time the guards of activities within the responsibility of a MARCA are also determined according to the process definition and the MARCAs are initialized with these guards through the Workflow Definition Tool.

Guards are logical expressions for significant events of activities of a MARCA like "start" and "terminate". MARCAs evaluate these guards with the messages that they receive to decide on their actions. As an example, the start guard of an activity handled by a MARCA can be the arrival of a document, say, "doc1" from site "S1" and a document "doc2" from site "S2". In this case, MARCA will start execution of this activity when this AND expression evaluates to true by the arrival of the mentioned documents. Clearly the guards are generated from the information given in the process specification. Similarly "terminate" guard of an activity handled by MARCA may require the transmission of a document, say, obtained from the in-house system to another MARCA. It should be noted that this transmission is realized through persistent queues to survive through crashes.

3 MARIFlow Security Services

Within the life cycle of a process definition an application in the company domain may prepare a message, then may pass it to local UMIA. UMIA, in turn, passes it to the corresponding MARCA. The MARCA sends the message to one or more other MARCAs according to the process definition. The receiver MARCAs pass the messages to their corresponding UMIAs, which pass them to the local applications. Thus, inter/intra company communications basically consists of communication sessions between MARCAs and between a MARCA and its UMIA.

The security requirements of the system can therefore be summarized as follows:

- Confidentiality of data transfer between MARCAs: The contents of a message, including the process description fields, should not be visible to anybody except the sender and the receiver.
- Confidentiality of data transfer between a MARCA and the corresponding UMIA, as above.
- Authentication and integrity for both levels.
- Signatures for some of the inter/intra-company messages, such as certain types of test and certification documents. Here, a message consists of a document that is passed in order to be stored.

A comprehensive set of security services has been developed and integrated into the system which are described in [2].

4 Failure and Exception Handling

MARIFlow system provides comprehensive set of measures for failure recovery and exception handling.

4.1 Recovery of MARCAs

When a site goes down, restarting the MARCA is under the responsibility of the Operating System's start up control. The site's start up control analyzes the persistent logs of the MARCA and start a new instance using these stable logs created before the site crash. However, there is need for a further mechanism to prevent any Operating System related problem.

In Mariflow, for each MARCA installed there is a background process at that site, called the "rescue process". The rescue process is responsible for monitoring the life time of the agent and checks the MARCA at specific time intervals through a predetermined socket. A thread of the MARCA listens to this socket and responds to the signals. If the MARCA does not respond to this process for a given period of time, the process starts sending signals more frequently. If the MARCA still does not respond, after sending a bunch of signals the process assumes that the MARCA is not functional. The two possibilities in this case are: the MARCA could be blocked or it could be dead. When the rescue process is unable to find the OS process that belongs to this MARCA (i.e., it is dead), it instantiates a new MARCA by the help of the persistent logs related with the state of the agent.

Otherwise if the MARCA is blocked, it is necessary to kill the old instance prior to installation of a new instance. Since the logs are persistent it is possible to recover the state of the MARCA killed and hence the site does not suffer from any inconsistencies.

For the described mechanism to work correctly it is necessary to make sure that rescue process stays alive. Therefore, just as the rescue process checks to see that the MARCA stays alive, the MARCA also checks to ensure that the rescue process stays alive by signalling the rescue process at predefined time intervals. It is MARCA who reinstantiates the rescue process when it dies.

4.2 Failure Handling

The hierarchical approach to failure handling described in [1] is implemented in MARIFlow system which allows for partially rolling back the workflow instance to the nearest point in process history tree where it is possible to restart the execution. When a sub-activity T fails, it is necessary to determine the impact of that failure on the ancestors of T by finding out the highest level ancestor that should be aborted. The details of this technique is given in [1].

4.3 Exception Handling

The MARIFlow system handles the following types of exceptions:

- Semantic exceptions occur when a deviation from the expected behaviour in the program logic is encountered. These are handled through the IF block.
- Exceptions Raised by the Communication Infrastructure: Various types of errors can be encountered during communication between two agents, or communication with a database system or a mail server. The communication system recovers from possible failures by retrying the operation when possible and informing the user program if the request cannot be issued.
- Exceptions caused by NON-VITAL Activities: It should be noted that when document flow is a part of a workflow system, more often than not, there will be a need to archive the documents. The transfer and archival of the documents may take considerable amount of time. Therefore a mechanism which allows the other activities in the system (that does not use these documents) to proceed without waiting these archival activities provides for better performance. Yet there should also be mechanisms to guarantee that the workflow instance will not terminate before the successful termination of all such activities. We use NON-VITAL activities suggested in [1] with some modification. Originally the NON-VITAL activities are defined to be those, whose failure does not effect the flow of the process. On the other hand we say that NON-VITAL activities are those that can be assumed to terminate as soon as they start but raise an exception when they fail. In this way a NON-VITAL activity does not delay the execution of other activities unnecessarily; yet their successful termination is guaranteed by the exception handling mechanism.

5 Conclusions

MARIFlow system implements a fully distributed inter-enterprise workflow system through cooperating agents. The system is developed within the scope of the European Commission supported INCO-DC 97 2496 MARIFlow project and is fully operational for industrial use.

References

1. Q. Chen, U. Dayal, "A Transactional Nested Process Management System", in Proc. 12th International Conference on Data Engineering, New Orleans, 1996.
2. A. Dogac, Y. Tambag, A. Tumer, M. Ezbiderli, N. Tatbul, N. Hamali, C. Icdem and C. Beeri, "A Workflow System through Cooperating Agents for Control and Document Flow over the Internet", Technical Report, Software Research and Development Center, METU, March 2000, http://www.srdc.metu.edu.tr/publications.html.
3. D. Hollingsworth, *"The Workflow Reference Model"*, Technical Report TC00-1003, Workflow Management Coalition, December 1996.

E-business: Key Issues, Applications and Technologies
B. Stanford-Smith and P.T. Kidd (Eds.)
IOS Press, 2000

An Agent-based Workflow System for Inter-enterprise Business Processes

Asuman DOGAC, Yusuf TAMBAG, Arif TUMER, Murat EZBIDERLI and Necip HAMALI
Software Research and Development Center, Faculty of Engineering
Middle East Technical University (METU), 06531 Ankara Turkiye
asuman@srdc.metu.edu.tr

Abstract. An agent-based inter-enterprise workflow system is developed within the scope of INCO-DC 97 2496 MARIFlow Project for automating and monitoring the flow of control and data over the Internet among different organisations. The main characteristics of the system are as follows: it has a completely distributed co-operating agent based infrastructure, provides a graphical workflow design tool as well as comprehensive monitoring facilities, handles exceptions and contains mechanisms for failure recovery. The system is also capable of automatically activating external applications (which may be inside company firewall) when necessary, keeping track of process information, and providing for the security and authentication of documents.

The MARIFlow system in comparison to the monolithic workflow engines and traditional client/server architectures, implements an architecture where the business processes are executed through distributed, cooperating agents interacting with one another over the Internet through messages. MARIFlow system is general enough to be applied to any application domain where data flow and invocation of activities follow a pattern that can be described through a process definition, however the example application is on maritime industry. The system is fully operational for industrial use.

1 Introduction

Workflow systems, in general, provide for declarative means for specifying the control and data flow among activities. Extensive research and development, both in the academia and in the industry have contributed to various aspects of the workflow system in making a mature technology. However in most of the systems, data flow is restricted to the parameters of the involved activities often disconnected from the description of the flow itself. In other words, the workflow management systems as used by the industry today, use a process centric approach. Thus, they lack a mechanism with which it is possible to define the source of data, control its flow over the net, and identify and possibly invoke the activities that make use of it. The workflow system that is developed as part of the MARIFlow project addresses these crucial issues with a special emphasis on the ease of use, maintenance, and customization to the needs of maritime industry.

In the MARIFlow system, the higher order process is defined through a graphical user interface, which is then mapped to a textual language called FlowDL. FlowDL allows indicating the source of the documents, their control flow and the activities that make use of these documents. A process definition in FlowDL is executed through co-operating agents, called MARCAs (MARIFlow Co-operating Agents), that are automatically initialized at each site that the process executes. The initiation of agents and monitoring facilities are managed through Java programs, which can be used by authorized users through Web.

MARCAs are responsible for handling the activities at their sites, routing the documents in electronic form, according to the process description, among other MARCAs, keeping track of process information by logging activities, and providing for the security and authentication of documents during communication.

The overall system should be immune to failures, and the processing power should be distributed such that it will not generate a serious bottleneck built around a single machine, which is accessed extensively. The participants of the workflow system should decide and behave on their own rather than being invoked or commanded by other programs in a centralized fashion. These requirements necessitate an agent-based architecture for the workflow system, where independent entities capable of completing complex assignments without intervention are used rather than tools that must be manipulated by a user.

In the MARIFlow system, the responsibilities of MARCAs in order to support the requirements given above are defined as follows:

- A MARCA receives messages through a persistent queue and evaluates them to decide what action to take. The persistent queue is necessary so that the agent does not lose its state after a program crash, site failure etc.
- It persistently stores the documents it receives. If the organization that the MARCA resides on has a firewall mechanism, it is also MARCA's responsibility to pass the documents to the in-house system, get the resulting documents from the system and forward them to the related agents as specified in the process definition.
- Process related information should be stored persistently for further monitoring purposes. Therefore it is MARCA's responsibility to direct related information to a database through its JDBC interface.

This paper is organized as follows: In Section 2 the overall architecture of the MARIFlow System is described. This section introduces the MARIFlow Cooperating Agents (MARCAs), the FlowDL workflow definition language, and monitoring of the workflow processes. The failure recovery and exception handling issues are discussed in Section 3. Finally, Section 4 concludes the paper.

2 The Architecture of the MARIFlow System

2.1 An Overview of the Architecture

Figure 1 gives an overview of the general architecture of the MARIFlow system. Each organization may have in-house applications inside a firewall protected from unauthorized access. MARCAs exist on a host machine outside the firewall relative to the site network. The MARIFlow agent informs the in-house applications when necessary through an internal process initiator and is responsible for sending and receiving documents and process related information through the firewall mechanism.

In MARIFlow an inter-enterprise workflow is defined graphically where the workflow designer specifies domains, tasks and process information which are used in building the process definition. The graphical representation of the process definition is mapped to FlowDL, the language used in MARIFlow, and from that definition the specifications for each MARCA is generated. The behavioral structure, obtained from the process definition for each MARCA, is then transmitted to the corresponding agent and MARCAs become available for distributed workflow management. It should be noted that this transmission along with the communication at instance level is realized through persistent queues in order to survive through system crashes and other problems.

2.2 MARIFlow Cooperating Agents: MARCAs

A workflow instance in MARIFlow is executed by co-operating agents called MARCAs. There is exactly one MARCA at each site participating to the workflow execution and it handles all the activities running at its site. At initialization, once in their life time, the sites download the generic MARCA template from a given URL. At compilation time the guards of activities within the responsibility of a MARCA are determined according to the process definition. Guards are special mechanisms based on intertask dependencies [1, 7]. They are logical expressions for significant events of activities of a MARCA like "start" or "terminate". MARCAs evaluate these guards with the messages that they receive to decide on their actions. In other words, guards inform a MARCA when to execute a certain activity. A detailed formal description of obtaining guard expressions from a given workflow specification is given in [4]. All of the information about guard structures is obtained from the process definition at the initialization phase of the MARCA.

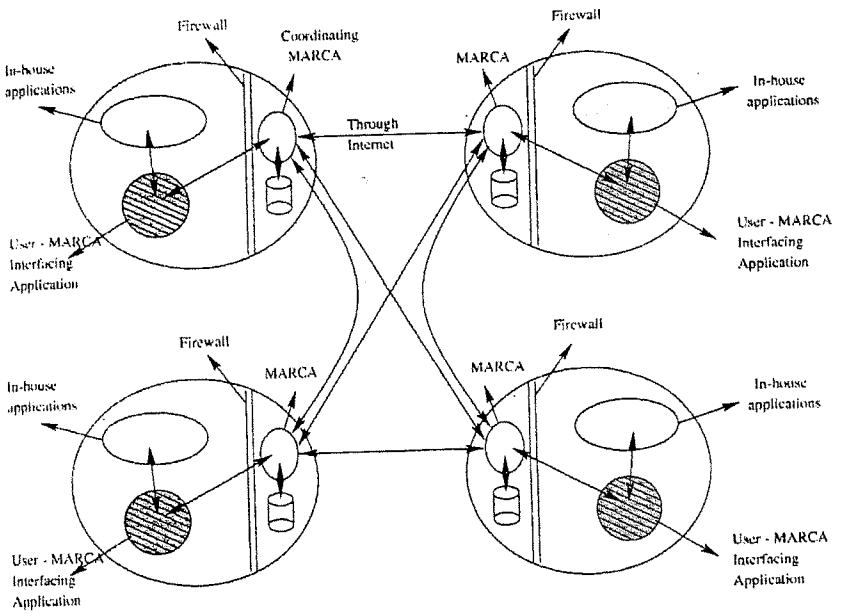


Figure 1. An Overview of the Architecture

A single MARCA residing on a host is capable of handling multiple workflow definitions and multiple instances of a given workflow at a certain time. All messages are differentiated by unique workflow id obtained from the definition and instance ids that are automatically assigned by the system for each instance generated.

MARIFlow agents communicate with each other through TCP/IP over the Internet. Network concurrent accesses to a single port is handled by Java's Net Package by assigning dynamic ports for each request. Simple message buffering and queuing are also provided by this package. These queuing facilities have been extended to persistent queue implementation and transactional agent communication for safe and consistent transmission.

MARCAs communicate with each other through an Agent Communication Language, which is specific to MARIFlow agents, hence they do not communicate with agents in the outside world. This is because communicating with outside world agents is not necessary within the scope of the work. During a session, each message is preceded by an activity identifier along with workflow and instance ids. This information is necessary to identify how a received document or message should be managed according to the activity identifier.

A MARCA sends a document and process related information inside the firewall through an e-mail. Since the content of the documents may be binary, they are encoded prior to the attachment by traditional mime base64 encoding used in e-mail messages. A program inside the firewall, named UMIA (User-MARCA Interfacing Application), processes the incoming mail and extracts the documents, process related information, workflow and instance ids, the activity that will use this document inside the firewall and displays this information through a GUI. The UMIA handles the tasks automatically or may let the user manage them through the GUI, according to its configuration. All the communication between the MARCA and the UMIA are handled through POP3 and SMTP protocols.

2.3 *Workflow Definition Language: FlowDL*

In MARIFlow system, a workflow process definition is given in FlowDL as a collection of blocks, tasks and other sub-processes as well as some explicit declarations and commands to specify Internet domain addresses, sources of documents, process specific information to be used for monitoring the document flow, and activities for further processing on the documents. The term *activity* is used to refer to a block, a task, or a (sub)process.

FlowDL contains several kinds of blocks, which are used to define different kinds of flow types in the process definition: the activities that run in parallel, in serial or under conditions etc. These blocks along with the declarations done in the process definition code define the whole workflow system. The block types encapsulate the workflow primitives defined in [5], which are sequential, AND-split, AND-join, OR-split, OR-join and repeatable task.

In MARIFlow, an inter enterprise workflow is defined graphically by using a definition tool. This tool allows the workflow designer to make the declarations, specify domains, tasks and process information. Afterwards, different types of blocks can be generated and added to the workflow by using the declarations in order to preserve consistency. The workflow definition is mapped to the textual FlowDL language. When this definition is parsed, the guards of each MARCA participating the system are generated and the agents are initialized with this information. The guards provide for the behavioral definitions of the MARCAs.

The advantages brought by block structured FlowDL language can be summarized as follows:

- The current workflow specification languages are unstructured and/or rule based, as noted in [6]. Unstructured languages make debugging/testing of a complex workflow difficult and rule based ones become inefficient when they are used for specification of large and complex workflow processes due to the large number of rules and overhead associated with rule management. FlowDL avoids these disadvantages through its block-structured nature.
- A block structured language confines the inter-task dependencies to a well-formed structure that in turn proves extremely helpful in generating the guards of activities for distributed scheduling of a workflow.

- Blocks not only clearly define the data and control dependencies among tasks but also present a well-defined recovery semantics through compensation for a failed or aborted block.

In addition to activities, there are also assignment statements in FlowDL which access and update workflow relevant data. In this way the workflow designer has the ability to assign values to variables. Note that these variables are used in conditional and loop blocks.

2.4 *Monitoring of Workflow Processes*

Each MARCA stores all the documents and messages it receives persistently in its database. Since Java is used in coding the MARIFlow system, a Java native JDBC interface is used for database connectivity. Consequently any database with a JDBC interface can be used by the MARCA.

The MARCAs also send a copy of the information they store, to the central monitoring database. The authorized users can track the flow of a process instance through a graphical user interface by giving the process instance identifier and the workflow definition it belongs to. The Web interface reads the process definition of the selected workflow from the database, and produces a graphical representation from that information, with different colors for each block type and with lines connecting the blocks showing the flow of data and messages. Also the interface reads the instance information from the central monitoring database and reflects it on the graph so that the user can see the instant state of the process instance on the screen, along with the finished, on going and yet to be started activities.

3 Failure and Exception Handling

Workflow processes are long-running activities consisting of many nested sub-activities. Many of the activities within the scope of a workflow instance may commit (i.e., make their changes permanent) during its execution and therefore when a workflow instance fails, already committed activities need to be undone to erase the effects of an unsuccessful instance. Compensation activities are used for this purpose to logically undo the effects of the activities in case of failures.

3.1 *Recovery of a MARCA*

When a site goes down, restarting the MARCA is under the responsibility of the Operating System's start up control. The site's start up control is to analyze the persistent logs of the MARCA and start a new instance using these stable logs created before the site crash. However, there is need for a further mechanism to prevent any Operating System related problem.

In MARIFlow, for each MARCA installed there is a background process at that site, called the "rescue process". The rescue process is responsible for monitoring the lifetime of the agent and checks the MARCA at specific time intervals. If the MARCA does not respond to this process for a given period of time, the process starts sending signals more frequently. If the MARCA still does not respond, after sending a bunch of signals the process assumes that the MARCA is not functional. A new MARCA is instantiated by the help of the information kept in the persistent logs related with the state of the agent. Note that prior to installation of a new instance, the old MARCA process is killed. Since the logs are persistent it is possible to recover the state of the MARCA killed and hence the site does not suffer from any inconsistencies.

3.2 Compensation

Effects of terminated activities of failed workflow instances in MARIFlow are undone through compensation activities. If there is a compensation activity for the whole block this one is used. Otherwise the compensation activities of the involved activities are used to roll back the block. When a parallel activity is to be aborted; the activities currently in execution in each branch receives a signal and the branches are compensated.

Since workflow processes are long-running activities aborting a workflow process instance for a failed activity is usually not desired. It is necessary to roll back the instance to a stable state and restart the execution possibly through an alternate track (roll forward). That is provided by the contingency blocks in the system.

3.3 Exception Handling

There is a need to handle the exceptions that may arise during the execution of the workflow process instances. An exception is an unusual event, erroneous or not, that is detectable either by hardware or software and that may require special processing. The type of exceptions handled by the MARIFlow System are as follows:

- **Semantic Exceptions:** Semantic exceptions occur when a deviation from the expected behaviour in the program logic is encountered like when a flight booking activity can not find any available seats in a flight. The alternative actions can be specified by testing the returned value by this activity within the scope of an IF block. A contingency activity may serve the same purpose by giving an alternative course of activities when an activity fails; however IF block also allows some conditions to be tested while specifying alternate actions.
- **Exceptions raised by the Communication Infrastructure:** Various types of errors can be encountered during communication between two agents, or the communication with a database system or a mail server. The MARIFlow communication system recovers from any possible failures, retries the operation when possible and informs the user program if the request cannot be issued. The errors can be categorized as: *Connection Errors, Communication Errors, Protocol Dependent Errors.*
- **Exceptions caused by NON-VITAL Activities:** Some activities in the workflow system may take considerable amount of time to finish successfully. A mechanism which allows the other activities in the system (that does not use these documents) to proceed without waiting these archival activities provides for better performance. Therefore we use NON-VITAL activities suggested in [2] with some modifications. NON-VITAL activities are those that can be assumed to terminate as soon as they start but raise an exception when they fail. In this way a NON-VITAL activity does not delay the execution of other activities unnecessarily; yet their successful termination is guaranteed by the exception handling mechanism.

4 Conclusions

In this paper we describe the architecture of a workflow system for document and control flow over the Internet realized through cooperating agents. The architecture provides for the declarative specification and automatic generation of the application rather than producing large amounts of application specific code by the help of block structured nature of the workflow definition language.

The system brings solutions to user interaction through the Java based Web interfaces,

architecture and it is more resistant to localized failures than that of centralized systems. The system also has an adaptable design and implementation so that the engine will run in different platforms ranging from a small set of personal computers to high-end workstations and mainframes by the universal programming language Java.

The architecture is general enough to be applied to any domain however the example application is provided for maritime industry and therefore some of the user interfaces are customized accordingly. Nevertheless the core of the system, workflow specification, communication details and database activities are suitable for any kind of domain definition thanks to the layered architecture.

The prototype system is developed within the scope of the INCO-DC 97 2496 MARIFlow project supported by the European Commission and is fully operational for industrial use.

References

[1] P. C. Attie, M. P. Singh, A. Sheth, M. Rusinkiewicz, "Specifying and Enforcing Intertask Dependencies", *Proceedings of the 19th VLDB*, 1993

[2] Q. Chen, U. Dayal, "A Transactional Nested Process Management System", *in Proc. 12th International Conference on Data Engineering*, New Orleans, 1996.

[3] A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Edtrs.), Workflow Management Systems and Interoperability, NATO ASI Series, Springer-Verlag, 1998

[4] A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, and M. Altinel, "Design and Implementation of a Distributed Workflow Management System: METUFlow", *in [3]*, pp. 61-91.

[5] D. Hollingsworth, "The Workflow Reference Model", *Technical Report TC00-1003, Workflow Management Coalition*, 1996, Accessible via: http://www.aiai.ed.ac.uk/WfMC/

[6] A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, and A. Wolf, "Report from the NSF Workshop on Workflow and Process Automation in Information Systems", *Sigmod Record*, Vol. 25, No. 4, pp. 55-67, December 1996.

[7] M. Singh, "Synthesizing Distributed Constrained Events from Transactional Workflow Specifications", *in Proceedings of the 12th International Conference on Data Engineering*, 1996.