リアルワールドOSとIoEワークフロー・アーキテクチャ
- IoE時代の自律分散協調型のワークフロー構築とマネージメントの為のオープン・プラットフォーム・アーキテクチャを目指して -

○出口弘　(東京工業大学大学)

Real World Operating System(RWOS) for IoE Workflow Management on the Internet
- Toward an Open Platform for Autonomous Distributed Cooperative Workflow on the Internet-

* Hiroshi Deguchi (Tokyo Institute of Technology)

Key Words

IoE (Internet of Everything), リアルワールドOS (Real World Operating System), ワークフロー (Work Flow), POEデータ (Point of Event Data)

1　第二次インターネット革命としてのワークフロー革命

1993年前後に商用インターネットがスタートして以降、社会の情報の流れや在り方は根底から変わった。このインターネット革命とも言うべき変化はグローバルなビジネス環境も大きく変化させた。これをここでは第一次インターネット革命と呼ぶ。第一次インターネット革命は、コミュニケーションの革命であった。UUCPネットワークでの電子メールや電子ニュースのネットワークであるinternetが、IP接続のThe Internetへと進化していった（以下インターネットは The Internet を示す）。この革命は、世界中の人々を結ぶ事で、コミュニケーションの有り様を激変させた、コミュニケーションの革命である。それはCPUのダウンサイジングにより、人々がパソコンやスマホを普通にコミュニケーションのツールとして用いる様になり、エンジニアの世界から市民の世界にまで広がっていった。結果として２０世紀を特徴付けていたマスメディアやTV放送等の巨大なコミュニケーションサービス（サーバ）は凋落し、Webやその上のブログからSNSまでボトムアップな情報発信が凄まじい勢いで広がった。これは決してコミュニケーションの島を作らせまいという明確なイデオロギーの下で発展してきたインターネットのコミュニティにより火をつけられた革命であり、そこではどこの国の政府も決してインタネットを切断することができないような分散的なアーキテクチャが確信犯的に採用された。

しかしこのようなボトムアップなコミュニケーション革命には、一つの想定外の誤算があった。情報発信は個へとダウンサイジングし、その通信のプラットフォームも特定の政府に決して制御されない自由度を持ったが、他方でコミュニケーションのプラットフォームとしてのSNSや検索サービスなどのプラットフォームサービスは、そのプラットフォーム財としての特質から強いロックイン特性を持ち、寡占化が進んだ。さらに様々な取引で生じるデータ(POE： Point of Eventデータ）としての顧客データの集積そのものが、サービス生産のためのの生産関数を構成する重要な要素として、従来の設備資本と労働に並ぶ或はそれ以上の重要さを持つ様になった。結果として、情報ネットワークサービス固有の階層的なプラットフォーム構造とその上に蓄積されるユーザの様々な活動イベントに付随する POE データの集積がこの新たなロックイン構造を引き起こした [Deguchi, 2004, 2014A]。 そこでは大量生産による収穫逓増メカニズムに依拠した独占とは異なる、新たな寡占、独占が生じた。

現在インターネットは、そこにつながるノードが爆発的に増大し新たな段階に入りつつある。このインターネットの新たなフェーズは、もののインターネット(IoT:Internet of Things)或は人もソフトエージェントも含めたあらゆるインテリジェントな膨大なノードがインターネットにつながるIOE:Internet of Everything)と呼ばれる[CISCO,2013]。だがこれは第一次のインターネット革命の極相としてのフェーズであると見なせる。人と人だけでなく、人とソフト、人ともの、ものとソフトのデータ交換が可能な高密度のネットワークが構築されることそれ自体は、第一次インターネット革命の必然的な帰結である。むろんその結果として生じる世界では従来できなかった様々なことが可能になり、その産業への影響は大きいと見積もられている。だが現時点で多く語られているのは、膨大なセンサー情報を利活用するクラウドビジネスであり、新たに取得が可能となる膨大な個人や社会のPOEデータを利活用しての巨大でロックインすることを目的としたビジネス構築への方向性が見え隠れしている。

むろんひと・もの・ソフトなどの膨大な数の自律的エージェントがインターネット上で相互作用するその相互作用の場が、インターネットのエッジであるという視点からは、CISCOによりFog Computingが提唱され、クラウドのみならずネットのエッジでの技術が模索されている[CISCO,2013]。またビジネスシステムの構築手法そのものも、大規模なウェブサービスの形を取るSOA(Service Oriented Architecture)からアジャイルで組み替え可能なマイクロ・サービスへとパラダイムシフトが生じつつある[pwc,2014]。

しかしこの領域では自律分散協調的なエージェントが、組織化されたタスクとしての一連のワークフローをインターネット上で実現する為の、ソフトウェアアーキテクチャの不在が顕著である。従来からのソケット通信や、CORBAでは、絶えずノードが入れ替わったり変化するシステムの中で組織化されたタスクを実行するプログラムを記述し、それをメンテナンスすることは難しい。またdockerなどの仮想化によるコンテナ技術も、それ自体は分散的なノード間のワークフローを構築するのを支援するものではない。

ここではインターネットに接続された人や機械やソフト等の知的エージェントをIoEエージェント或は自律分散協調型エージェント(Distributed Autonomous Cooperative Agents:DAC Agents) と呼ぶ。このDACエージェントが、インターネット上で様々な組織化され

た活動を、既存の企業や組織の壁を越えて可能とすることに、次のインターネットの変革の本質がある。DACエージェント間で可能となる膨大で多彩なワークフローの構築は、既存のビジネスモデルの延長上で捉えられない変化を社会にもたらす可能性を持つ。インターネット上の自律分散協調型の知的エージェントが単にコミュニケーションする或はエージェント間のコミュニケーションを通じて既存のビジネスモデルがスケールアップされたり範囲を広げたりするだけでないより根源的な変化が惹起されることが予想される。

このインターネット上の膨大なDACエージェントがもたらす多彩なワークフローの変革を本稿では第二次のインターネット革命と呼ぶ。この第二次のインターネット革命では、従来比較的緩い時定数で局所的かつ疎に接続して運営されてきた、人や機械やソフトウェアの行うタスクを連結したワークフローが様変わりすることが予想される。相互に接続され組織化された活動を遂行する、DACエージェントにより実現されるワークフローの爆発的な増大とそれが我々の社会経済経営システムに及ぼす影響こそが第二次インターネット革命の本質がある。それがもたらすのは、工場や家庭からコミュニティまで様々な社会的な場にあるIoEノードを連結するシステムのボトムアップな革新である。ビジネスからノンプロフィットまでDACエージェントからなる様々なワークフローが可能とする新たな社会・経済的機能により、社会に新たな多様な付加価値の創成が可能となるだろう。それは従来の会社組織に所属して会社組織が形成する付加価値の配分を得る、給与所得者としての労働者の在り方やそれを取り巻く、組織的にオーガナイズされた人間活動システムの在り方をも徐々にではあるが変えていく可能性がある。

膨大なIoEノードが相互に結合して、組織化された活動をするときにその組織化された活動をどのように認識し、モデル化し、それを遂行し、評価するかというのが課題となる。それは人間だけがワークフローの要素であった時代から続いている課題でもある。人間によってタスクが遂行される場合のワークフローは、通常の社会組織のワークフローでありしばしばプロジェクトとも呼ばれる。その分析には、PERTやCPMなどの手法が古くから開拓されてきている。

IoE時代には、センサーやアクチュエータ等の「もの」或は「ソフトウェアエージェント」をタスク遂行の自律的主体と見なした上で、それらが結びついて「秩序だったタスクの遂行とその過程でのデータ処理」が課題となる。この「ネットワーク上で結びついたタスク遂行可能な資源に対しての秩序だったタスクの遂行とその過程でのデータ処理」を行うことは容易ではない。実際に現在のプログラミング言語やUML(Unified Programming Language)のようなモデリングのフレームワークは、ここで述べているような「ネットワーク上で結びついたタスク遂行可能な資源に対しての秩序だったタスクの遂行とその過程でのデータ処理」を定式化するモデル化の原理も、プログラミングの処理系も提供していない。このようなシステムを開発するためには、組込み系に近い個別のネットワークプログラミングとして開発するしか現在は方法はない。しかしそれでは膨大なネットワーク上の資源を結びつけ効果的に機能させる為のワークフローの絶えざる構築と再構築のプロセスを支えるインフラとしてはあまりにも脆弱である。他方でこのようなシステムを、基盤となる標準化のもとで、中央に巨大なデータベースを構築し、その上でビジネスロジックを記述する様な形で実装することで対処できるのだろうか。現在のIoTに関するビジネスソリューションは、このようなデータベースを基軸とした処方箋で設計・開発されることが多い。Industry 4.0はこのような視点から工場からサプライチェーン迄の広範な物作りのシステムを規格化して標準化仕様とする戦略である。しかしこのような中央集権的な設計方法論による、物作りや産業のIoE化、システム化は、日本のものづくりの根幹にある、現場のケーパビリティ（能力開発）を重視して、絶えざる改善を可能とするものづくりとは相容れない。中央集権的なデータベース（クラウド）を基軸としてIoE時代のサプライチェーンを含む産業システムのプラットフォームを構築することになれば、それは日本の産業の根幹となる数多くの中小企業の工場を、フランチャイズのチェーン店の様に、中央が全ての情報を集約し、末端はオペレーションを行う取り替え可能なシステムとしてしまう危険性がある。これは日本の産業基盤を弱め、国の産業競争力の根幹を揺るがすことになる。

次節では我々は、IoEワークフローの構築のための階層的なアーキテクチャモデルを示し、その上でデータフローソリューションという分散型のデータ処理の方法と、プロジェクトプログラミングという、ネットワーク上で複数のタスクが半順序関係で結ばれたワークフロー（これをプロジェクトと呼ぶ）を設計・実行するためのプログラミング手法を提案し、それらを統合し実世界のIoEノード上での組織化されたタスクの遂行が分散的に行われるプロセス自体を支援するリアルワールドOSという枠組みを併せて提起する。

2　IoEワークフロー構築の為の階層モデル

本節では、インターネット上での自律分散協調型の分散データ処理の新たな枠組みを提起する。膨大なネットワーク上のDACエージェントからなるIoEノードが相互に結びつき形成されるであろう無数のワークフロー（以下DACエージェントからなるワークフロー或は単にDACワークフロー：DACWF）を、どのように技術的に構築しどのようにマネージするのか、またそれがどのような社会経済的な可能性をもたらすのかに関する領域透過的なビジョンを示す。

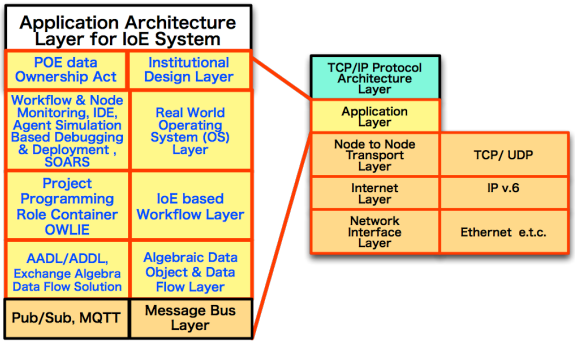


fig.1　DACWF構築ための階層アーキテクチャモデル

そのために、まずfig.1に示される様な、ネットワー

ク上でIoEワークフローを構築しマネージするために参照するための階層モデルを提起する。この階層モデルはTCP/IPの４階層モデルのアプリケーション層を、IoEワークフローの構築の視点から再階層化したものであり、DACワークフローに対する一つのアーキテクチャ提案となっている。

そこではまずネットワーク上のノード間を結ぶメッセージバスの層を、TCP/IPのネットワーク上のアプリケーション層に設定する。更にその上にノード間のデータフローとそこでのデータ構造を規定する層を設定する。更にその上にネットワーク上でのワークフローをネットワークプログラムとして実現する層を導入する。

【メッセージバス層】

DACエージェントとして個々のタスクを遂行する主体の物理的実態を示す場合は、IoEノード或は単にノードと呼ぶことにする。ノードの実態はスマホを持った人のこともある。また何らかのセンサーやアクチュエータなどのマシンエージェントがIoEノードとして機能する場合も、ソフトウェアエージェントの場合もある。このノード間のメッセージ通信を、TCP/IPのアドレス指定とをする事無く、N:Nで行えることは、柔軟なノード間通信を実現する上では必須となる。我々はアプリケーション層上にノード間のメッセージ交換のためのバスとなる層を想定する。ここではそのために、Publish/Subscribe(Pub/Sub)のプロトコルとその実装モデルとしてのMQTT(Message Queuing Telemetry Transport)<http://mqtt.org/>プロトコルを利用している。このプロトコルでは、ノードからノードへの通信は、トピックを指定したデータをMQTTサーバへ送る（ Publishする）ことと、必要とするトピックをMQTTサーバにSubscribe登録しておくことでなされる。MQTTサーバは、トピックのついたPublishされたデータを、そのトピックをSubscribe登録してあるノードへと転送するというデータ転送の役割を果たす。それゆえデータの送り手と受け手がこのトピックをキーとして、MQTTサーバを経由してデータをやり取りすることができる。

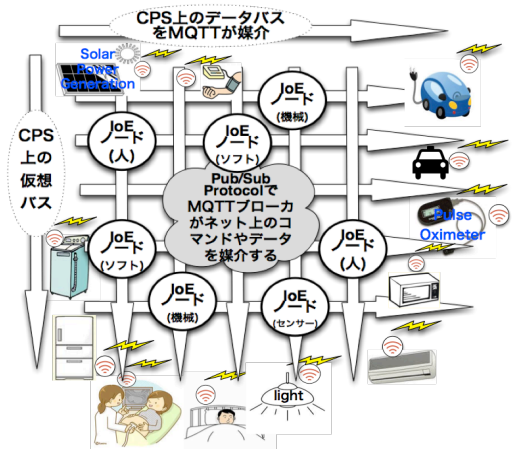


fig.2　仮想バスのPub/Sub プロトコルモデル

このPub/SubPub/Sub プロトコルによるノード間でのデータ転送は、HTTPプロトコルによるデータのやり取りに比べて、3G通信下では、受け取りで100倍、送出で10倍程度効率の良い事が知られている<http://public.dhe.ibm.com/software/dw/jp/websphere/connectivity/ms_mqtt_ws/MQTT_MessageSight_seminar_2.pdf>。

なおメッセージバスでのプロトコルとして、我々はPub/Subを想定し、ここではその実装モデルとしてMQTTを年とうい於いて議論を行うが、Pub/Subプロトコルの実装モデルはMQTTだけではなく、Java Message Service(JMS)やXMPPなど幾つかのものがあり、また今後リアルワールドOSの機能に必要な拡張が行われる必要もある。

【データフロー層と代数的データオブジェクト】

計算等のコンピュータの資源が希少であった時代には、データは静的に畳み込まれたデータ構造として扱われてきた。このような方法にRDBとXMLDBがあるが、例えばXMLで会計的な状態を表現したXBRL((eXtensible Business Reporting Language))を考えてみよう。会計システムは、上流で発生する様々なトランザクションをダブルエントリーで記述し、それを様々な形で加工する計算体系である。そこには上流部の取引伝票のような出発点があり、そこから簿記的な計算により導出された一連のデータ群が簿記の計算体系となる。そこには例えば会計測定の粒度を変えてアグリゲートしたBSやP／Lなどの諸帳票も含まれる。これらの計算結果を一つの静的なデータとして表現するのが、XBRLの方法である。XBRLに限らず、様々なビジネスデータは、何らかの上流のイベントにより発生し、それが様々に加工されるデータフローを定義して、その中で扱われている。このデータフローのそれぞれのタスクのところでは、上流部のデータを加工し下流のデータを生成する計算のフィルターとして扱えることとなる。

これに対し、現在のビジネス情報処理の主流の方法は、データフローを陽に示すのではなく、計算結果を何らかの形で静的なデータ構造として収納する方法となっている。

これを簡単な事例で示す。顧客表、 製品表と、 それに注文表を上流とし、例えばそこから買掛金伝票と、出庫表を計算するデータ処理を考える。従来のデータ処理の枠組みでは、まず問題に対するE-R図などでの分析を行う(fig.3)。しかしこのような例ではより永続性の高い顧客表、製品表がEntityとして扱われ、顧客のデータと製品のデータを含む形で生成される注文表は、Relationとして扱われ、それを元にRDBが構築され、その上でビジネスロジックが書かれ、SOAとしてのウェブ上でのサービス構築がなされるのが今日のウェブソリューション或はデータベースソリューションと呼ばれるものの実態である(fig.4)。しかし実際に様々なデータの扱いで、EntityとRelationの区別に論理的な妥当性があるかというとはなはだ疑問がある。数理的にはそれらをデータから区分する方法は無い。更にこのようなソリューションでは、データの種類やそこでの様々な計算が頻繁に組み替えられる状況化では、システム構築が複雑さに絶えられない。これに代替する新し情報処理スキームが要求される所以である。そこでは仮にマイクロサービスにサービスを分解するとしても、そのマイクロサービスの単位をどのように構築して、またそれをどのように組み上げて全体サービ

スを構築するかで様々な可能性がある。我々が採用するのは、代数的なデータ表現を用いて、データ間の変換を代数的なデータに対する関数型のオペレーションとしてのその仕様記述を行い、この変換をマイクロサービスの基本単位（タスク）とし、それらのタスクを結びつけたワークフローをひとまとまりのサービス単位とする考え方である。これをデータベースソリューションに対して、データフローソリューションとここでは呼ぶことにする。

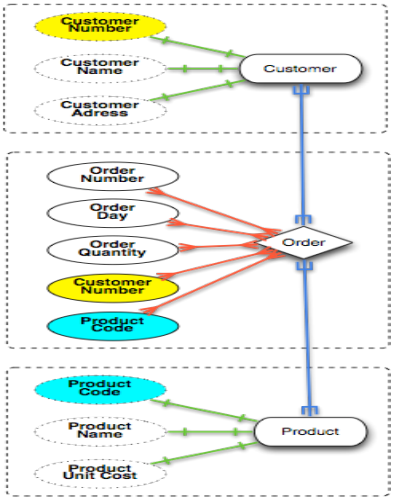


fig.3　E-R図による取引分析例

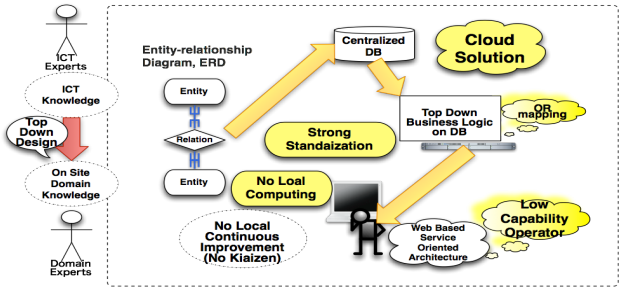


fig.4　E-R図に基づくDBソリューション構築

データフローソリューションの考え方では、まずデータを代数的に表現して、その代数的に表現されたデータ間の変換のフローを確定する。その上でそのデータフローをワークフローとして実現する枠組みを構築する。

そのためにデータの抽象的な表現の為の代数的基盤として、交換代数とデータ代数を導入する。交換代数は簿記を抽象化した代数系で、通常借方、貸方に分けてテーブル形式で表現される複式のフロートストックの状態記述を、代数的に表現し公理的にこれを特徴づけたものである[Deguchi,2004;Mattessich,2007]。これに対しデータ代数は通常のレコード形式のデータを部分代数として表現したものである。これらの代数系を用いる事で、簿記形式のデータとレコード形式のデータが代数的に表現可能となり、その間の変換を代数的なオペレータを含む集合論的な仕様記述で容易に行うことが可能となる。

具体的なフィルターの機能仕様は、交換代数、データ代数と集合論の表現を用いて記述される。マイクロ・サービスの基本単位としてのフィルターは、集合論的に構築された関数として明確に仕様記述がなされる。

このようにプログラムでではなく、数理的に一つ一つのマイクロ・サービスの仕様記述ができる事は、サービスの仕様記述の頑健性と言う視点からは極めて重要となる。この仕様記述を実際のデータ処理のプログラム・コードに落とす為に、DSLとしてのAADL/ADDL及びその統合開発フレームワークとしてのFalconSeedを既に我々は構築しダウンロード可能な形で提供している〈www.soars.jp/〉。AADL/ADDLでは、フィルターの記述は、内包的に行われ、ほぼ集合論的に記述された仕様と対応する形でプログラムの記述が可能となっている。このデータフローを先の例で示すとfig.5のようになる。

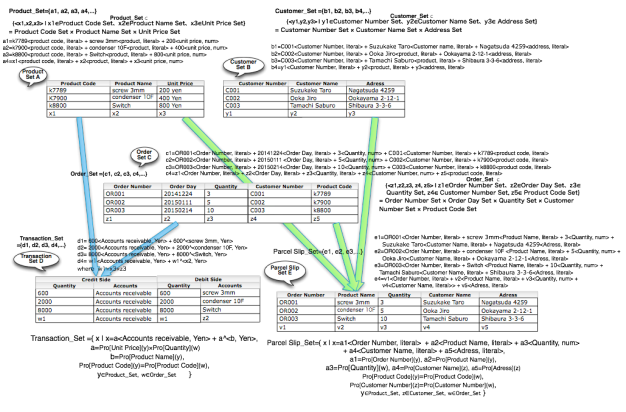


fig.5　代数的に表現されたデータフローの例

このデータフローを少し詳細に見る。表は2次元のシンプルなテーブルで次の様に表現されている。個々の表での行は一つのレコードを示す。

顧客テーブル

Customer Number	Customer Name	Adress
C001	Suzukake Taro	Nagatsuda 4259
C002	Ooka Jiro	Ookayama 2-12-1
C003	Tamachi Saburo	Shibaura 3-3-6
y1	y2	y3

製品テーブル

Product Code	Product Name	Unit Price
k7789	screw 3mm	200 yen
K7900	condenser 10F	400 Yen
k8800	Switch	800 Yen
x1	x2	x3

注文テーブル

Order Number	Order Day	Quantity
OR001	20141224	3
OR002	20150111	5
OR003	20150214	10
z1	z2	z3

このレコードとその集合からなる表形式のデータの集合は、次の様に代数的に表記可能である。

（1）Product_Set={a1, a2, a3, a4,...}
Product_Set ⊂
{<x1,x2,x3> | x1∈Product Code Set、x2∈Product Name Set、x3∈Unit Price Set}
= Product Code Set×Product Name Set×Unit Price Set
a1=k7789<product code, literal> + screw 3mm<product, literal> + 200<unit price, num>
a2=k7900<product code, literal> + condenser 10F<product, literal> + 400<unit price, num>
a3=k8800<product code, literal> + Switch<product, literal> + 800<unit price, num>
a4=x1<product code, literal> + x2<product, literal> + x3<unit price, num>

（2）Customer_Set ={b1, b2, b3, b4,...}
Customer_Set ⊂
{<y1,y2,y3> | y1∈Customer Number Set、y2∈Customer Name Set、y3∈ Address Set}
= Customer Number Set × Customer Name Set × Address Set

b1=C001<Customer Number, literal> + Suzukake Taro<Customer name, literal> + Nagatsuda 4259<address, literal>
b2=C002<Customer Number, literal> + Ooka Jiro<product, literal> + Ookayama 2-12-1<address, literal>
b3=C003<Customer Number, literal> + Tamachi Saburo<product, literal> + Shibaura 3-3-6<address, literal>
b4=y1<Customer Number, literal> + y2<product, literal> + y3<address, literal>

（3）Order_Set ={c1, c2, c3, c4,...}
Order_Set ⊂
{<z1,z2,z3, z4, z5> | z1∈Order Number Set、z2∈Order Day Set、z3∈ Quantity Set, z4∈ Customer Number Set, z5∈ Product Code Set}
= Order Number Set × Order Day Set × Quantity Set × Customer Number Set × Product Code Set
c1=OR001<Order Number, literal> + 20141224<Order Day, literal> + 3<Quantity, num> + C001<Customer Number, literal> + k7789<product code, literal>
c2=OR002<Order Number, literal> + 20150111 <Order Day, literal> + 5<Quantity, num> + C002<Customer Number, literal> + k7900<product code, literal>
c3=OR003<Order Number, literal> + 20150214<Order Day, literal> + 10<Quantity, num> + C003<Customer Number, literal> + k8800<product code, literal>
c4=z1<Order Number, literal> + z2<Order Day, literal> + z3<Quantity, literal> + z4<Customer Number, num> + z5<product code, literal>

これらの顧客データ、製品データ、注文データを上流のソースデータとすることで、そこからは、注文データと製品データから仮払いの取引伝票が計算により生成される必要がある。更に製品を出庫するための配送伝票が生成され、それが印刷され荷物に添付される必要がある。

注文取引伝票テーブル

Credit Side		Debit Side	
Quantity	Accounts	Quantity	Accounts
600	Accounts receivable	600	screw 3mm
2000	Accounts receivable	2000	condenser 10F
8000	Accounts receivable	8000	Switch
w1	Accounts receivable	w1	z2

出庫データテーブル

Order Number	Product Name	Quantity	Customer Name	Adress
OR001	screw 3mm	3	Suzukake Taro	Nagatsuda 4259
OR002	condenser 10F	5	Ooka Jiro	Ookayama 2-12-1
OR003	Switch	10	Tamachi Saburo	Shibaura 3-3-6
v1	v2	v3	v4	v5

これらは上流部にあたる顧客データ、製品データ、注文データから、代数的なオペレーションとして計算され、結果も同様に代数的に表記できる。下記にその計算と計算結果を示す。

（4）Transaction_Set ={d1, d2, d3, d4,...}
Transaction_Set ={ x | x=a<Accounts receivable, Yen> + a^<b, Yen>, a=Pro[Unit Price](y)×Pro[Quantity](w), b=Pro[Product Name](y),
Pro[Product Code](y)=Pro[Product Code](w),
y∈Product_Set, w∈Order_Set }
d1= 600<Accounts receivable, Yen> + 3^<screw 3mm, Pieces>
d2= 2000<Accounts receivable, Yen> + 5^<condenser 10F, Pieces>
d3= 8000<Accounts receivable, Yen> + 10^<Switch, Pieces>
d4= w1<Accounts receivable, Yen> + z3^<x2, Pieces>
これを価格表示にすると下記になる。
d1= 600<Accounts receivable, Yen> + 600^<screw 3mm, Yen>
d2= 2000<Accounts receivable, Yen> + 2000^<condenser 10F, Yen>
d3= 8000<Accounts receivable, Yen> + 8000^<Switch, Yen>
d4= w1<Accounts receivable, Yen> + w1^<x2, Yen>
where w1=x3×z3

（5） Parcel Slip_Set={e1, e2, e3,...}
Parcel Slip_Set={ x | x=a1<Order Number, literal> + a2<Product Name, literal> + a3<Quantity, num> + a4<Customer Name, literal> + a5<Adress, literal>,
a1=Pro[Order Number](y),
a2=Pro[Product Name](y),
a3=Pro[Quantity](w),
a4=Pro[Customer Name](z), a5=Pro[Adress](z)
Pro[Product Code](y)=Pro[Product Code](w),
Pro[Customer Number](z)=Pro[Customer Number](w),y∈Product_Set, z∈Customer_Set, w∈Order_Set }
e1=OR001<Order Number, literal> + screw

3mm<Product Name, literal> + 3<Quantity, num> + Suzukake Taro<Customer Name, literal> + Nagatsuda 4259<Adress, literal>

e2=OR002<Order Number, literal> + condenser 10F <Product Name, literal> + 5<Quantity, num> + Ooka Jiro<Customer Name, literal> + Ookayama 2-12-1<Adress, literal>

e3=OR003<Order Number, literal> + Switch <Product Name, literal> + 10<Quantity, num> + Tamachi Saburo<Customer Name, literal> + Shibaura 3-3-6<Adress, literal>

e4=v1<Order Number, literal> + v2<Product Name, literal> + v3<Quantity, num> +v4<Customer Name, literal>> + v5<Adress, literal>

データベースソリューションでは、E-R図から出発しDBを作成、ビジネスロジックを記述、ユーザインタフェイスを開発するがそこには殆ど現場知とのコミュニケーションが無い。データベースソリューションは現場の情報処理をERPパッケージの様に規範化する傾向が強く、現場の『改善』を認めないが結果としてエクセル等による現場コンピューティングが氾濫するという現状がある。これに対し、我々の提唱するデータフローソリューションは現場知からの情報処理を可能とするフレームワークである。交換代数とデータ代数は、企業の簿記データとレコード形式のデータを抽象化し、代数的に表現し、それをデータオブジェクトとして利活用できる様に定式化したものであり，組織で流れるデータの多くの部分がこの二つの代数形式で表現できる。

データフローソリューションでは、「データ構造は計算である」というテーゼのもとで、上流のデータから必要なデータはフィルター型の計算モジュールにより次々と計算される形で得られると想定する。このようなデータフローアプローチでは、システムの改善は容易である。これを示す為にfig.6にデータフローとその変更の例を示す。

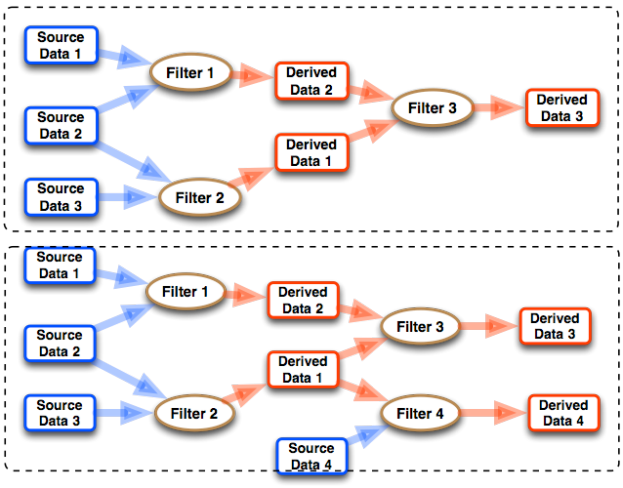


fig.6　データフローとその変更の例

データフローの下流に「フィルター４」や「データソース４」を付け加えても、データフローとしては既存の計算に影響しない。また更にデータの上流部の改変も含めて新たな計算方式を導入する際も、従来のデータフローを並列して残しつつ比較して、徐々に新しい方式へと移行することがかのとなる。このようにデータフロー型のシステムは、個々のフィルターをマイクロサービスの最小単位（タスク）としつつ、全体として一つのワークフローで実現されるべきデータ処理を設計することができる。

旧来のデータベースを中心としたデータ処理パラダイムに対して、我々は「データ構造＝データ処理フロー」という視点に立っている。そこで提示されるのはデータを上流から下流に向けて計算フィルターというマイクロサービスで関数的に変換しながら流していく、データフロー型のソリューションである。最小単位のマイクロサービスは、変換フィルターという基本タスク単位によって構築される。より大きなサービス単位とは、データフローを構成するフィルターの適用の半順序の流れから構築される。このデータフローの半順序構造は、我々が提供するデータフローソリューションのためのフレークワークであるFalconSeed上のマクロとして実現できる。これは単一のマシン上で実装する事も可能だが、他方で多くのデータ処理サービスは、ネット上の環境として実現される。従来はサーバ中心のデータベースソリューションとして実装されているところを我々は次に述べる様に、ワークフロー層でこれをインターネット上でのIoEノードの自律分散協調的なワークフローで実現することができる。会計計算で例をとれば、上流の伝票や諸取引データから新たなレコードや取引形式のデータを組み立てながら最終的に期間内フロー情報としての損益計算書やストック情報としての損益計算所を構築することになる。そこでの計算のプロセスを構成するマイクロサービスは、データ代数間の変換、データ代数と交換代数との相互変換、交換代数間の相互変換のフィルター群からなる。ここで重要なことは当該のマイクロサービスは、その組織的な担当部門に取って容易に理解可能な形で構成されているという点である。fig.3-5と関連する諸表で示した先の例では、出庫担当部門は、むろん顧客テーブル、製品テーブル、注文テーブルから出庫データテーブルを作ることについてのサービスの意味も構造も理解していなければならない。同様に会計担当部門は製品テーブルと注文テーブルから、複式の簿記記述である注文取引伝票テーブルを構成するやりかたを理解していなければならない。またこれこそが組織に於ける分業の在り方となる。デーてベースソリューションやその結果としてのERPといったビジネスソリューションでは、全てのビジネスロジックを標準化して中央で統制する設計となっているが、このような設計は組織的な分業が持つ現場力を削減し、現場を単なるオペレータと見なす危険を孕む。実際にある種のフランチャイズ型のビジネスでは、フランチャイジーのビジネスプロセスは全てセンターで設計され、現場はオペレーションマニュアルに従った作業を情報システムの指示に基づいてするにすぎなくなる。このようなビジネスモデルの設計は、ビジネスを粗なコンポーネントに分解して、付加価値の低いコンポーネントを人件費の低い地域へとオフショアするといったコンポーネントビジネスモデルの系譜の一つと見なすこともできる。フランチャイジーなどのオペレーション現場は低付加価値のコンポーネントと見なされ、低いケーパ

ビリティのワークで運営され、それらはやがて機械に置き換わっていくことが想定されている。このようなサービスプロセスのデザインでは、現場からの改善も現場からの新たなスピンアウトや暖簾分けによる展開も許されない。これは日本の物作りを考えた時に、極めて重大な、またマクロにはこれは社会成層の分解をもたらし、消費を主導するのみでなく、これからのサービス経済の主力となるべき高いケーパビリティを持った多様な人材の排出を妨げ、所得のみならずケーパビリティのデバイドを社会にもたらしかねない。

更にデータフロー層では、データの静的な構造としての規格化無しにデータをコンバートすることで不要な規格化なしでのデータの扱いを我々は主張する。我々はデータ構造を静的にではなく、動的に計算されるデータフローそのものとして把握する。この視点からは、複雑に構築された静的なデータ構造で表現される「規格」は無用の長物となる。むろん個々のデータ単位、我々の体系ではデータ代数と交換代数で表現されるレコード型と簿記型のデータはそれぞれ固有の構造を持つ。しかし簿記はそれ自体が複式の記述の体系でありそれは交換代数によって定式化されている[Deguchi, 2004]。様々な情報をレコードデータとして表現する方式はむろん欄の構成や単位の選択等に任意性はあるが、それらは論理的に構築されたものであり、対象を同じくするものであれば、データ測定の粒度の差を除けば必ず変換可能である。他方で例えばXBRLのように会計的なデータの全体をXMLの形でツリー上に構造化する規格は、我々の立場からはデータフローとして可能な計算のストリームを一つの静的な構造の中に畳み込んだにすぎずそれを規格として共有する意味は少ない。同じデータを構築するデータストリームは関連するデータの構築も含め企業によって様々に実装され得る。問題は必要なデータを個々のマイクルサービス単位で現場の理解のもとで構築でき、それが更に全体として動的なデータフローを構築できることである。その上で、貸借対照表や損益計算書のような対外的に規格化されたデータが導出できればよいのである。

このビジネスタスクの中核を構成するデータフローの計算に加え、センサーデータの利用や様々な制御やそれらに関しての人間とのやり取りを等を含む様々なIoEノードのネットワーク上で自律分散的で協調的な組織化された(Organized)タスクをプログラミングする手法として導入されるのがプロジェクトプログラミングという枠組みとそれを実現するOWLIE(Onsite Workflow Language for Internet of Everything or
Own Way Life on Internet of Everything)である。

【ワークフロー層】

ワークフロー層は、IoEノードが個々に遂行するタスクとしてのマイクロ・サービス間を結ぶワークフローを定義し、それを実行する層である。一般にネットワーク上のノードを結ぶプログラミング技法としては、ソケット通信によるものや、CORBAによるものが知られている。しかしこれらはノードの頻繁な組替え等のノード間のデータフローの変化に追従して、ワークフローを実現するプログラムを書き換えるためにはあまりにそのプログラミングは煩雑となる。このインターネット上でのノード間の通信を含む、自律分散協調的なノード間プログラミングは如何にして可能となるのかに答えるのが我々のプロジェクトプログラミングの枠組みである。ここではこのプロジェクトプログラミングと呼ぶ新しい自律分散協調型のネットワークプログラミングの方法と、そのための専用言語を導入する。

【ステージ概念と三種の並列性】

ワークフローをコンピュータ上でどのように表現するかについては、ペトリネットによるワークフローの分析など従来から幾つかの研究がある。また人間を対象としてのワークフローの研究として、プロジェクトマネージメントの領域での多くの研究がある。 生産管理の領域では、順序づけられたタスクを一つのジョブとして、複数のジョブの実行の最適化を課題とする。他方でいわゆるプロジェクト管理の領域では、CPMやPERTではより複雑な半順序関係のタスクの遂行とそこでのクリティカルパスなどのボトルネック解析が行われる。 特に、プロジェクトのスケジューリングに関しては、クリティカルパス法（CPM)や、PERT（Program Evaluation and Review Technique）がよく知られている。これらはいずれも複数のタスクからなる一つのワークフローをある種の半順序関係をタスク間の実行順序を示す矢印などで記述し、ワークフロー全体の遂行にとってのボトルネックパス（クリティカルパス）などの解析を行う。また資源割り付けのガントチャートなどの導出も行う。

プロジェクトプログラミングでも個々のタスクが半順序関係で結ばれたプロジェクトを対象とする。ただし我々の問題関心は、インターネット上に散在するIoEノードでDACエージェントに依って個々のタスクが実行される形で、全体としてのプロジェクトを如何に組織化して遂行するかのプログラムの実行制御にある。

これに対して導入されたのがステージ概念による分散実行制御である。ステージ概念はマルチエージェントシミュレーションでの役割概念に基づいたエージェントの役割遂行活動のモジュール化に於いて、役割遂行の実行制御のための概念として、エージェントベースシミュレーション言語SOARSで初めて導入された概念である〈www.soars.jp〉。ステージは、離散時間システムで、１ティック（δt)の単位時間を複数のスライスに分けるスライスとして定義される。SOARSではシミュレーションの１ティックは複数のステージに分割され、ステージ間では因果的な前後関係を持つが、同じステージに属する役割の遂行は複数のエージェント間でどの順序で行っても良いという並列性をマルチエージェント間でのタスクの遂行について保証するために用いられる概念である。プロジェクトプログラミングではこのステージ概念を、タスクの半順序関係として定義されるプロジェクトのワークフローに対して拡張した概念として導入する。シミュレーション上では、タスクの遂行は１つのコンピュータ上でのプログラムとして行われ、マルチエージェントの役割遂行の並列性は、疑似並列にすぎない。しかしモデル設計上でエージェントの並列動作を織り込んだモデル化をステージ概念は可能とする。プロジェクトプログラミングではこのステージ概念を、実世界のDACエージェントによるIoEノード上で実行する、実世界のマルチエージェントシステムのモデル化と実行制御に関して拡張する。

そのためには、SOARSでのマルチエージェントプログラミング同様に、実行すべきタスク＝ステージとしてその順序関係を記述する。ただし実世界でのタスクの遂行では、順序関係はプロジェクトのPERT図同様に、半順序関係となる。

図７はプロジェクトプログラミングに於けるステージの構成例である。ここで左のワークフロー１(WF1)では、線形順序で３つのステージ表現されているが、WF２ではステージ２と３がステージ間が並列となる半順序としてステージ間関係が表現されている。なおワークフローで表されるステージ間関係は単位期間に遂行される一つのプロジェクトを示すが、これは１回で終わる場合もあれば、ある種の制御過程のようにリピートされ繰り返す場合もある。また後述する様に複数のワークフローがそれぞれ１回だけ実行され、そのクラスター全体を制御する場合もある。

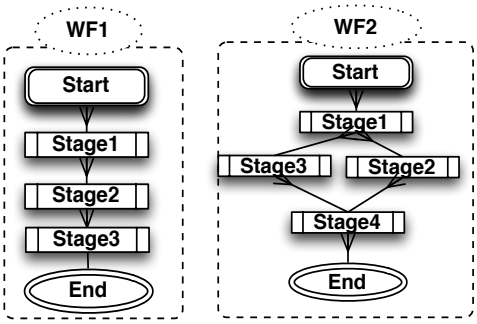


fig.7　プロジェクトプログラミングのステージ例

ステージ概念を用いることで、リアルワールドOSでは３種類の実世界の並列性を扱う事が可能となる。一つは、あるステージを遂行するIoEノードが複数ある場合の、その複数のIoEノードの遂行の順序を問わないという意味での並列性である。これは例えばデータの収集のステージで、複数の温度センサーからデータを取得する際にどのセンサーから最初にデータを送られてもよい、或は学生から数学のレポートを回収するというステージで、学生がどの順番でレポートを出してきても無関係という意味での並列性である。さらにこれとは別にタスクを遂行するステージ間の並列性概念がある。これは図７のWF２に於けるステージ２とステージ３の関係である。例えばここでステージ２が国語のレポートの回収のタスクで、ステージ３が英語のレポートの回収のタスクとする。ステージ４で全体のレポートの採点とその平均点を計算するステージとすると、ステージ２と３は並列に遂行することができる。更に第三の並列性はWFそのものに関する並列性である。これは例えばWF１とWF２などの複数のワークフローが並列に実行される場合の並列性にあたる。この並列性は、プロジェクトの並列性であり、これもまた実世界のワークフローの並列実行で重要な概念となる。例えば、WF２が後述するような何らかの製品の製造プロセスを表すワークフローであるとしよう。複数の製品をロットとして生産するということは、その製品の個数だけのワークフローを実行するということであり、これを並列に実行できれば全製品の製造までの時間が短縮される。同様にWF１とWF２がマンション建築に於ける二つのタイプの部屋のそれぞれの内装工事の工程を表すものとしよう。そのマンションでは、WF１タイプの部屋が２０部屋、WF２タイプの部屋が３０部屋あったとして、全体の内装工事はWF１を２０、WF２を３０遂行することで完了する。このワークフロー自体の遂行の並列性もまた重要な課題となる。このワークフローの遂行の並列性の前提に、タスクへの資源の割当という課題が存在する。この資源の割当という課題を含むワークフローの集合に対するスケジューリング問題については後述することとし、ここではまずステージに対応して実世界のIoEノードの資源が固定的に割り当てられているプロジェクトについて考察する。

【資源が固定的に割り当てられたワークフロー】

図８は温度センシングと照度センシングに基づく実世界での自律分散協調型のタスクの遂行の事例である。ここではセンシングされた温度データと照度データから平均温度と最小照度を計算し、それに基づきエアコンのコントローラと照明装置のコントローラをそれぞれ起動させるコマンドを発行し、また同時にそこで用いられるエネルギーを計算しエネルギー会計報告を作成し関係者に送るというワークフローを表す。このワークフローはリピート型で一定時間事に繰り返される。

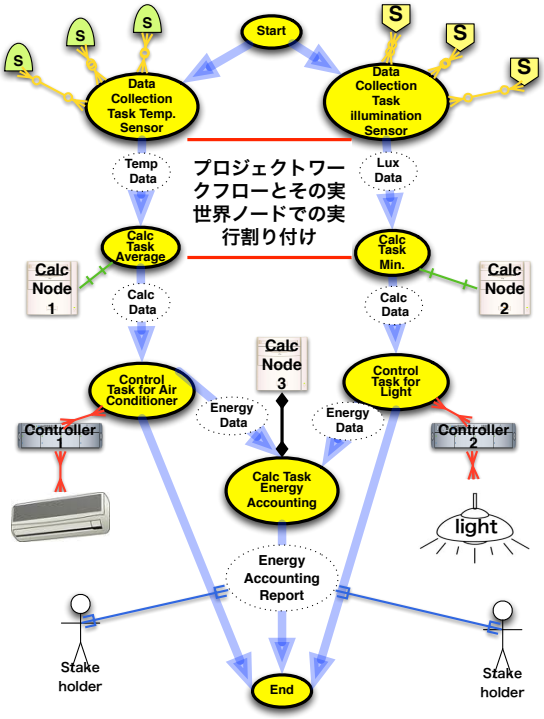
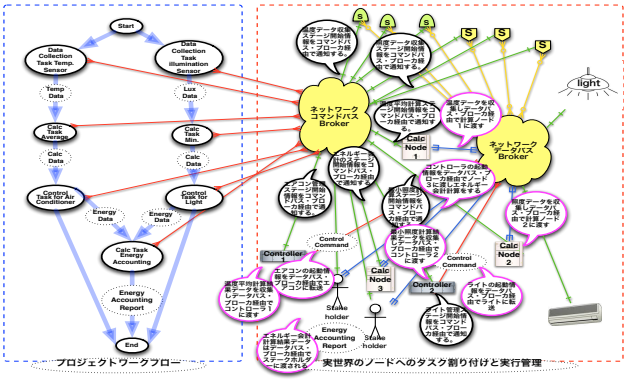


fig.8　センサリングに基づく実世界での自律分散協調型のタスク遂行の事例

このワークフローでは、黄色の実践のノードがステージを表す。図ではそこにセンサーや計算ノード１−３、コントローラノード１、２、エアコン、ライト、ステークホルダーなどの実世界のDACエージェントが紐づけられている。従ってこのワークフローでは、資源の割当は既に唯一に与えられている。これを実際に実世界のIoEノードの組織化された自律分散協調的なタスクとして実現する為には、プロジェクトプログラミン

グに基づく起動させるには、今どのステージを実行すべきかに関するステージ情報がプロジェクトプログラムのマネージャからブローカにコマンドに関するトピックをつけて送られ、それをサブスクライブしたノードがそこでのタスクを実行し、実行終了後それをマネージャに通知する或は一定条件でタスクが打ち切られ次のステージへの移行がなされるなどのステージ管理メカニズムが必要となる。タスク間での組織化されたワークフローの遂行のためにはこれら実行順序の並列性を含む制御と同時に、実行の結果得られたデータの転送の為の枠組みが必要となる。これも我々はタスクの終わったノードが結果を適切なトピックをつけてパブリッシュして、それを次にタスクを遂行するノードがサブスクライブするという形で実装する。図１でネットワーク上の仮想バスとして想定しているのは、このコマンドのバスとデータのバスである。

このネットワーク上でのタスク制御とデータ制御を論理的にコマンドを媒介するブローカとデータを媒介するブローカを区分して（現実には区分する必要はない）表示したのが図９である。



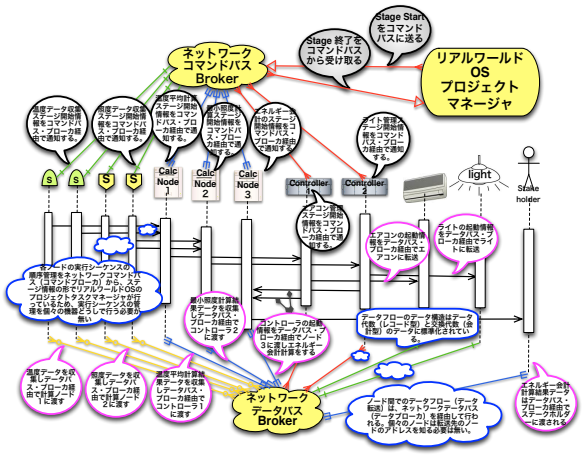


fig.９　センサリングに基づく実世界での自律分散協調型のタスク遂行の事例

ただしここでシーケンス図のように見える部分はデータの流れを示しており、センサーからの計算ノードへの転送の順序関係は意味を持たない。またこのようなネットワーク上のノード間プログラムをシーケンス図で制御しようとすると、非常に複雑なものとなり設計自体が大変な負担となりそのメンテナンスも容易ではない。

先に我々はデータフローソリューションについて述べた。図５で示されたデータのフローは、ワークフローとして表現することでIoEノード上での計算資源や人によるデータ入力、結果の表示などと結びつけられ、実世界のワークフローとして具体的なデータフローの計算が遂行される。ここではデータのセンシング（入力確認）ステージと、計算ステージ、結果の表示ステージの３ステージモデルとしてワークフローを表現している。

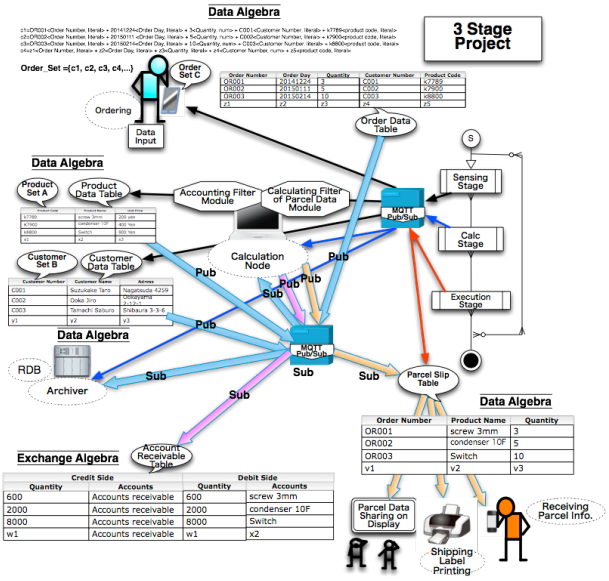


fig.10　データフローソリューションのDACエージェントからなるワークフローとしての表現

【ロールコンテナ】

プロジェクトプログラミングでは実際に実世界のDACエージェントで必要なタスクを実行する為に、そのタスクを遂行するためのネイティブな実行環境と、それをプロジェクトプログラミングのステージモデルやブローカを経由してのデータのやり取りを支援する環境に分離し、後者を支援するミドルウェア上で前者を実行するコンテナ型のアーキテクチャを我々は導入する。

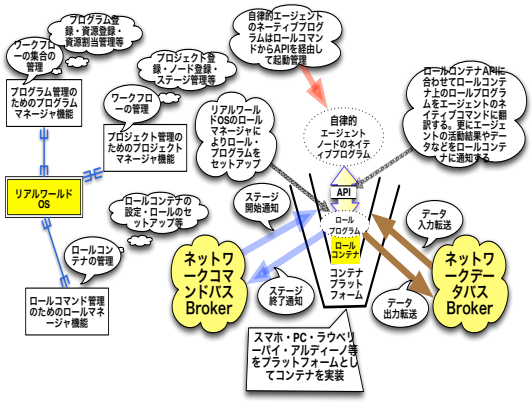


fig.11　ロールコンテナによるDACエージェントのIoEノード化

このプロジェクトプログラミング対応のミドルウェアを既存の様々な環境に導入することで、個々のIoEエ

ージェントの多様性をロールコンテナ上のAPIで吸収し様々なDACエージェントをプロジェクトプログラミング対応のIoEノードとすることができる。図１１はこのロールコンテナのアーキテクチャを示している。

ロールコンテナは、DACエージェントをプロジェクトプログラミングの中で役割遂行を行うためのノードとして用いる為にIoEノードの上に実装されるミドルウェアである。プロジェクトプログラミングでは、各ステージでの役割は、SOARS同様に、ステージ名と、役割記述のIF Thenルールで記述される（αバージョンは直接関数名を記すがβバージョンではSOARS同様のステージ記述になる予定）。その役割の実質的な意味はAPIを経由して隠蔽されたDACエージェントのネーティブなコードで記述される。現在のシステムではロールコンテナはJavaで記述され、それを実装するコンテナのプラットフォーム環境ではJavaが動くことが求められる。ファルコンシード上で記述された計算フィルターは、Jロールコンテナと一緒に梱包されJARファイルとして配布することもできる。またロールコンテナを実装したシステム上で、PythonなどのJava以外の言語で記述されたコードを実行することもできる。またタブレットなどに実装されたコンテナ環境から、人に何らかのメッセージを送ったりメッセージを取得すると言った使い方もできる。更にロールコンテナはその上のDACエージェントのネイティブなコードや活動を隠蔽する機能も持つ。ブローカ上でやりとりされる情報はステージ情報や役割に関するもので、それが何を実装するかはネットワーク上からは隠蔽されるサンドボックスモデルを提供している。

ロールコンテナに類する概念は、Tom Catのようなウェブコンテナや Dockerのようなコンテナ型の仮装化技術など幾つかある。これらはいずれもある種のミドルウェアではあるがそれぞれ考え方は異なっている。我々がリアルワールドOSで用いるコンテナ概念は、インターネット上の各ノードに実装されることで、IoEエージェントの実態とそれがノードで果たす役割を分離する機能を持つミドルウェアであり、様々な実態を持ったIoEエージェントが一つのワークフローの中で協調的に働くための共通のフレームワークを与えるものとなる。

現実の人間組織でも、様々な人が組織内の役割を担い、それは属人的でない形で役割取得と役割遂行という形で実現される。役割がワークフローの中で定義され、それを実施する担当者は、役割を自身の行為にブレークダウンする必要がある。しかし役割をマネージするという視点からは、役割がどのような行為パターンで実現されるかは、外から見えない形で隠蔽されているほうがよい。プロジェクトのワークフローからは、どのタイミングでその役割が遂行され、それに必要な資源や情報が割り当てられていることが重要で、個々の実施方式を役割に対して定めるのは別の課題となる。

もう一つプロジェクトプログラミングで重要なことが、プロジェクトがフローチャートではないという認識である。プロジェクトを構成する個々のタスクは、ある条件で実行され、終了すると次のタスクへ移行する。プロジェクトのワークフローの中では、個々のタスクの失敗によるワークフローの失敗はあっても、ワークフローそれ自体は、半順序の遂行順序に従ってタスクを実行する枠組みであり、そこには、フローチャートの様なIf thenのような条件分岐は存在しない。逆に言えば、人間の組織はプロジェクト型のワークフローと言う形で、マルチエージェント（この場合は主に人）の間の複雑で組織化されたタスクの遂行を行ってきた。そこで経験的に蓄積されたワークフロー概念では、ワークフローを構成する個々のタスクの内部では様々なIf Thenの分岐があるが、それをタスクとしてまとめたものは、失敗は別とすると開始と終了、タスクに必要なデータや物資の取得、結果の適切なところへの報告や引き継ぎなどを基本的な外部へのインターフェイスとするだけである。この人間の組織的なワークフロー概念の特色を我々はプロジェクトプログラミングとして、実世界のDACエージェントからなる組織化された協調分散的タスク遂行に援用した。それゆえワークフローで記される実世界のDACエージェントのタスク遂行は、いわゆるフローチャートによる記述より遥かに見通しのよい構造を持つ。

【ワークフローの遂行支援環境としてのリアルワールドOS層】

プロジェクトプログラミングで記述されるワークフローの遂行は、単にプロジェクトプログラミングの実行環境をワークフローのコントロールマネージャとして用意するだけではない。コンピュータのOSの発展史を見ると、それは当初モニターとしてメモリー上にプログラムを読み込み実行するところからスタートして、プログラムを作成する環境、資源管理をする環境等様々な要素を付け加えて発達してきた。同様に実世界のIoEノードでのタスクの組織化された遂行を支援する為の様々な機能が総合的に提供される必要がある。リアルワールドOSはその総合的な支援環境として位置づけられる。リアルワールドOSが支援するのは、実世界のDACエージェントの組織化されたタスクの遂行だけではない。実世界で自律分散的に遂行される諸ワークフローに関してはその遂行のプロセス制御のみならず、多くの支援課題がある。 プロジェクトプログラミングの実行支援環境には、先に述べた複数のワークフローに対する資源の割当によるスケジューリング問題も含まれる。またプロジェクトプログラミングの設計から実装に至るまでのエージェントベースシミュレーションを用いたパーシャルデプロイメントという新しい実装支援の方法もある。

【パーシャルデプロイメント】

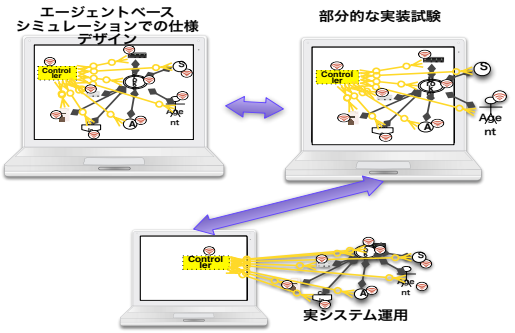


fig.12 パーシャルデプロイメントによるデバック

図12はパーシャルデプロイメントによるプロジェクトプログラミングのデバックとデプロイメントの流れを示したものである。プロジェクトプログラミングでは、実世界のノードでの複雑なデータのやり取りが行われる。その正当性を示すことは従来のプログラムの正当性理論の範囲を超える。しかしDACエージェントの相互作用の一部或は全部をエージェントベースシミュレーションでモデル化してやることで、従来とは全く異なる実世界でのプロジェクトプログラミングに対する正当性の検証が可能となる。エージェントシミュレーションは意図した動作をモデルが行っているかの検証のために用いることができる。また例えば何万人もの人間が災害時にデータを送り、それを用いて災害支援する様なシステムをプロジェクトプログラミングで構築する際には、災害時の人間の送るデータをエージェントシミュレーションでモデル化してやることで、シミュレーションモデルが現実の人間に代わりデートをブローカに送り、全体システムの挙動を確認するということもできる。同様のことは、センサーデータを集め何らかの計算により判断を行い、その結果制御を行う様なワークフローに関するプロジェクトプログラミングでも可能となる。センサーのデータを異常な値も含めエージェントベースのシミュレーション側でエミュレートしてそれをパブリッシュして現実のアクチュエータの挙動や制御計算の妥当性を検証する、或は逆に現実のセンサーのデータに対して制御計算やアクチュエータの挙動をシミュレートするなど、部分的に現実世界のDACエージェントが、部分的にエージェントベースシミュレーション上でのエージェントが合わさって一つのワークフローを実行することでそのワークフローの妥当性を検証する枠組みが我々の提案するパーシャルデプロイメントである。

【ワークフローのPDSサイクル支援とマルチワークフローに対する資源割当と実行スケジューリング】

既に我々は、プロジェクトに関して、資源の割当がその実行に必要な場合がある事を指摘した。

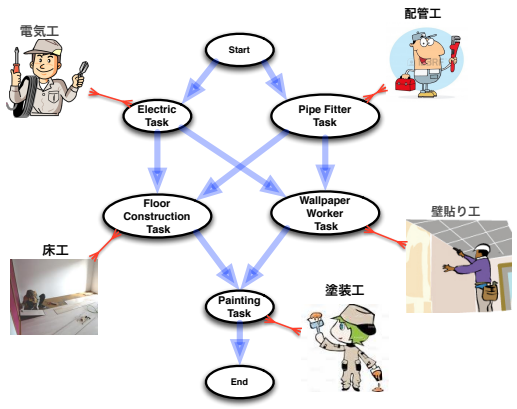


fig.13　簡単な内装工事のワークフロー

図１３は簡単な内装工事のワークフローである。ここでは電気工や配管工、床工、壁塗り工、塗装工等がタスクを実行する資源となる。内装工事の部屋が５部屋あり、電気工が一人の場合、電気工のタスクは順々に各部屋で行われる。つまり５つのワークフローそれぞれの資源がタスクに割り当てられることで実行される。ここで課題となるのは、実行順序のステージ制御ではなく、実行のためのマルチワークフローへの資源割当のスケジューリングとなる。このように複数のワークフローのクラスター（ロット）に対して、スケジューリングという形でプランニングに関する支援も実世界でのワークフローの遂行には重要な役割を果たす。更にワークフローを構成するタスクが異なる組織に属すこともある。図１４は高齢者の入院の介添えと入院中の自宅とペットの管理を示すワークフローだが、ここではペットの面倒と自宅のメンテナンスは外注であると仮定している。このようなワークフローではワークフローの遂行や、そのスケジューリングのみならず、そこでのサービス原価計算も課題となる。

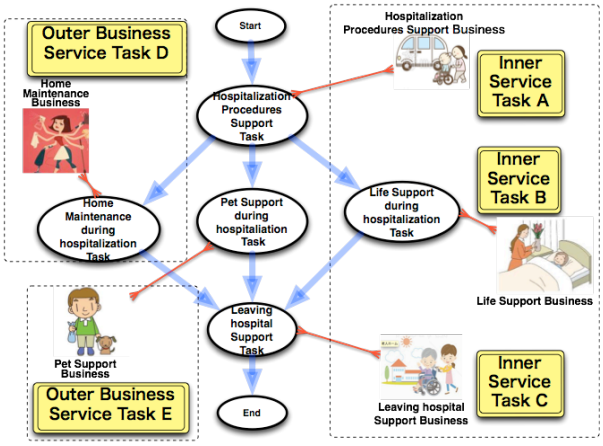


fig.14 複数の組織を跨がるワークフローの例

このようにリアルワールドOSが行うワークフローの支援では、ワークフロー及びその集合（クラスター或はロットと呼ばれることもある）に対して、そのデザインとスケジューリング等を含むプラン(Plan)のフェーズ、ワークフローの遂行(Do)のフェーズ、更にワークフローの遂行結果やコストや利益の計算等のモニタリング(See)のフェーズの全体に渡り支援を行うことが求められる。即ち実世界OSでのDACエージェントが資源割当も含めて実現するワークフローのPDSサイクルの全体に対する支援がリアルワールドOSに求められることになる。

【物作りとリアルワールドOS】

IoTではIndustry4.0がしばしば議論の俎上にのぼる。そこではサプライチェーン全域にわたる規格の統合やERP的なシステムの導入による生産システムの刷新が課題であると主張される。しかし既に述べた様に、我々のリアルワールドOSのアーキテクチャは、現場単位でのマイクロ・サービスを基盤とするアーキテクチャである。図１５は銅板と鉄板の切削加工とそれらのプレスと塗装という４つのタスクからなる簡単な部品の製造工程を示している。個々のタスクでは原料或は仕掛品をマテリアルとして投入しそれに加えて、その工程固有のファブリケーションサービスを投入することで結果として製品或は次のステップでの仕掛品が製造される。 またファブリケーションサービス自体は、労働と機械の稼働とエネルギーを投入として生産される。図１６はそれらの関係を示し、これが簿記的なダブルエントリーの表現で記述できることを示してい

る。

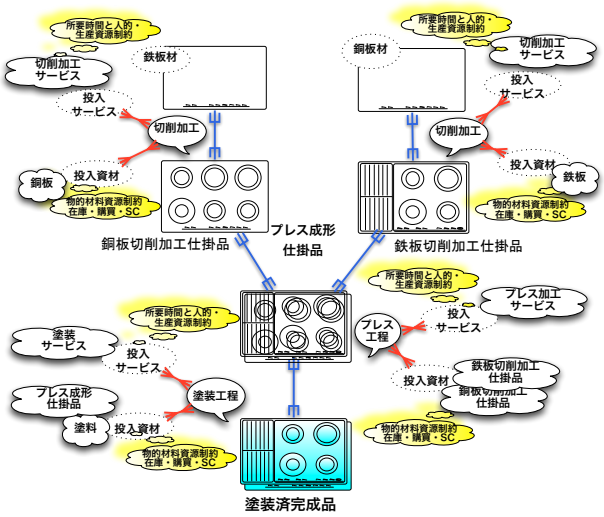


fig.15 簡単な製造プロセスのワークフロー

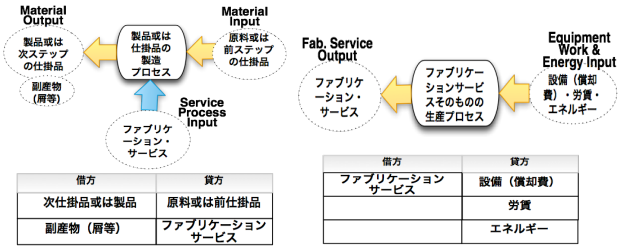


fig.16 製造タスクでの投入産出の構造

図１５を構成する個々のタスクを交換代数で表現すると次の(A1)-(A4)のように示される。

（A１） 鉄板切削加工仕掛品製造取引

x[鉄板切削加工仕掛品製造]
=1<鉄板切削加工仕掛品、個、*、*>
+ 5<鉄くず、Kg、*、*>
+ 20^<鉄材、Kg、*、*>
+ 2^<切削加工サービス、時間、*、*>

これは鉄板材20Kgを原料として、切削加工を２時間行うと、鉄板切削加工仕掛品１個と副産物として鉄くず５Kgが生成されることを示している。

（A２） 銅板切削加工仕掛品製造取引

x[銅板切削加工仕掛品製造]
=1<銅板切削加工仕掛品、個、*、*>
+ 2<銅くず、Kg、*、*>
+ 8^<銅材、Kg、*、*>
+ 1^<切削加工サービス、時間、*、*>

これは銅板材8Kgを原料として、切削加工を1時間行うと、銅板切削加工仕掛品１個と副産物として銅くず2Kgが生成されるという取引を示している。

（A３） プレス成形仕掛品生産取引

x[プレス成形仕掛品製造]
= 1<プレス成形仕掛品、個、*、*>
+ 1^<鉄板切削加工仕掛品、個、*、*>
+ 1^<銅板切削加工仕掛品、個、*、*>
+ 1^<プレス加工サービス、時間、*、*>

これは、鉄板切削加工仕掛品１個と銅板切削加工仕掛品１個を用いて、それに対してプレス加工を１時間行うとプレス成形仕掛品が１個製造されることを示している。

（A４） 塗装済み製品生産取引

x[塗装済み製品製造]
=1<塗装済完成品、個、*、*>
+ 1^<プレス成形仕掛品、個、*、*>
+ 2^<塗料、Kg、*、*>
+ 1^<塗装サービス、時間、*、*>

これは、プレス成形仕掛品１個に塗料２Kgを原料として、塗装サービス１時間行うと、塗装済完成品が１個製造されることを意味する取引を交換代数で示したもの。これらの関係から図１７のような部品展開表(BOM)も求められる。

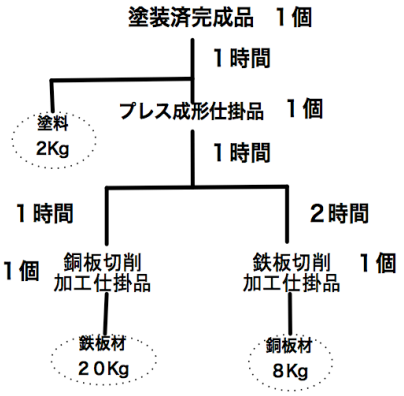


fig.17 BOM(Bill Of Materials:部品展開表)

このA1-A4の４つの工程の各々では、原料或は仕掛品を中心とした生産加工イベントがそれぞれ実物勘定を用いた代数的多元簿記によるトランザクションとして記述される。これは、POEデータとして製造タスクを認識し、それを複式の多元簿記で記述した形となっている。この記述は、１）マテリアルや仕掛品の投入、２）マテリアルを加工や利用するサービスの投入、という二種類の投入項目と、産出項目としての３）製品（仕掛品）の生成と４）副産物の産出という項目からなる取引として記述される。

その上で実際にマテリアルや仕掛品を実際に加工するサービスは時間単位で計測されるサービス投入として扱われ、そのサービス自体を遂行するのに必要な機械や人員やエネルギーなどの資本投入やその資本財を利用したサービスの時間制約などの産出関係の取引は別途記述され管理される。即ち、原料等の物質の投入に関する情報を記述したタスクのトランザクション記述と、それに対する加工サービスの部分を区別して記述する。

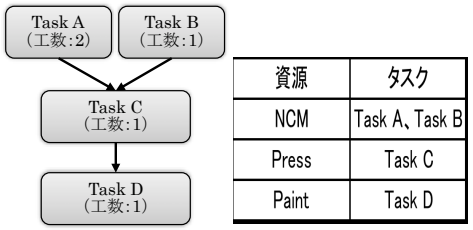
この事例では、１）切削加工サービス、２）プレス加工サービス、３）塗装サービスが製造タスクの中で時間単位で投入される加工サービスタスクとなる。次のB１-B３では、それらのファブリケーションサービスの単位時間辺りの内訳を示す取引を記述する。A1-A4で示した様なマテリアルに対するファブリケーションサービスの投入では、必要な時間数のサービスが投入されることになる。この事例では、 鉄板切削加工仕掛品製造では２時間、銅板切削加工仕掛品製造では１時間、切削加工サービスが投入されている。

【生産計画情報の抽出】

ここからMRPで調達計画を策定することになる。しかし一般には、この調達スケジュールは、生産スケジューリングを考慮に入れておらず、いわば無限生産能力仮定の下での計画となる。分かりやすい例として、図１で塗装済み完成品を１０００個基準日時に納品することを考える。BOMからの展開では、納品の１時間前までにプレス成形仕掛品が１０００個と塗料２０００Kgが必要とされる。さらにプレス加工に必要な日数から、その１時間前に、銅板切削加工仕掛品と鉄板切削加工仕掛品が１０００個ずつ用意される必要がある。更に銅板の切削加工に１時間、鉄板の切削加工に２時間かかることから、プレス加工の開始の１時間前に銅板8000Kgが、２時間前に鉄板20000Kgが用意される必要がある。しかしこの調達計画は、もし利用可能な切削加工の為のマシニングセンターが１台しかなければ根本的に組み直しが必要となる。鉄板版と銅板版が用意されても、まずその加工に延べ３時間かかるからである。更に１０００個であれば延べ３０００時間かかることになる。仮にプレス機械も１台とし、銅板切削加工と、鉄板切削加工の仕掛品ができ次第、プレス加工を順次行うとすると、３時間に一組ずつ切削加工仕掛かり品セットができることから、板材の切削加工を始めて３０１時間後には、３時間に１時間稼働させたプレスでのプレス成形仕掛品の製造も終了する。塗装プロセスでは塗装工がやはり１人として、全ての塗装が終了するのは最初の切削加工がスタートしてから３００２時間後となる。これはそれぞれの工程で提供されるファブリケーションサービスB１−B３で、１組の資源（塗装工、プレス加工機、切削加工機）だけが提供されるという仮定があるからである。逆に必要なだけ加工サービス資源が提供される状況では、２０００台の切削加工機と１０００台のプレス加工機と１０００人の塗装工が用意されれば、4時間で全ての作業は終了する。現実に用意される加工資源の下では、上限と下限の間で実際のスケジューリングはなされる。従って無限資源仮定の下でのMRPは必然的に過剰な在庫を抱えることになる。このようにBOMから構成されたMRPによる資材調達計画は、しばしば過剰な在庫をもたらす。その大きな原因は、生産機械やサービス提供に関する人員などの資源の制約にある。生産工程では原料や仕掛品とは別に、生産機械の割当や人員の割当等の時間単位での資源割当のスケジューリング計画が必要となる。BOMに基づくMRP的な資材の調達計画だけでは加工機械や労働資源等の、加工サービス遂行の為の資源制約によって生じる資源割当のスケジューリングの課題に答えることはできない。

ルネックとなる資源に注目する。切削、プレス。塗装の３つのサービスに対して、それぞれ、切削加工装置、プレス加工装置、塗装工がボトルネックとなる資源となり、その供給のタイミングが全体の工程のスケジュールとなる。

下記ではやや抽象的にワークフローとタスクの関係を示す。前述の例と同様に４つのタスクからなる製品を製造するワークフローを考える。その上でこの製品３個を１ロットとしてこれを製造するためのスケジューリングを考える。図１８はこの４つのタスクのワークフロー構造と、そこでの個々のタスクを遂行することのできる資源を示している。資源はNC工作機械がTask　A,Bに用いられ１台、プレス機はTask　C用で１台、塗装機械はTask　Dに用いられ１台とする。ここでは労働は制約条件から省いてある。



資源	タスク
NCM	Task A, Task B
Press	Task C
Paint	Task D

fig.18　４つのタスクからなる製造工程とタスク遂行用資源としての１台づつのNCM,Press,Paintマシン

【動的スケジューリング】

製品の製造工程のワークフローに対してロット単位で具体的にスケジューリングを行う為には、ワークフローを構成する個々のタスクを遂行する為に必要な資源をそれぞれのワークフロー毎に順次割り付けていく必要がある。そこで具体的にこの３個のロット生産問題でのスケジューリングの割り付けを考える。ファルコンシードが提供する動的スケジューリングのモジュールでは、複数の異なったディスパッチルールを適用することで機械装置などの資源をワークフローに割り当てることができる。結果はガントチャートとして示される。

上記の部品生産を例にとった事例を示す。３つの製品の製造ワークフローはそれぞれP１、P２、P３で示されるものとする。ワークフローを構成するタスクは、A,B,C,Dで示され、例えばP1:AはワークフローＩにタスクAが割り当てられたことを示すものとする。

表１は最大工数のタスクを優先割り付けするディスパッチルールによってスケジューリングを行った結果の資源毎のワークフロー上のタスクへの割当を示すガントチャートである。工数２のタスクであるAに優先的に工作機械が割り当てられている。 表２は逆に最小工数のタスクを優先した場合のガントチャートを示している。

表１最大工数タスク優先割付スケジューリング

資源＼時間	0	1	2	3	4	5	6	7	8	9	10
NCM	P1:A	P1:A	P2:A	P2:A	P3:A	P3:A	P1:B	P2:B	P3:B		
Press								P1:C	P2:C	P3:C	
Paint									P1:D	P2:D	P3:D

表２最小工数タスク優先割付スケジューリング

資源＼時間	0	1	2	3	4	5	6	7	8	9	10
NCM	P1:B	P2:B	P3:B	P1:A	P1:A	P2:A	P2:A	P3:A	P3:A		
Press						P1:C		P2:C		P3:C	
Paint							P1:D		P2:D		P3:D

表３ワークフロー優先&最大工数タスク優先割付スケジューリング

資源＼時間	0	1	2	3	4	5	6	7	8	9	10
NCM	P1:A	P1:A	P1:B	P2:A	P2:A	P2:B	P3:A	P3:A	P3:B		
Press				P1:C			P2:C			P2:C	
Paint					P1:D			P2:D			P3:D

これに対して表３では、同一ワークフローへの割当を優先し、その上で次に最大工数タスクを優先するデ

ィスパッチルールでのスケジューリングの割当結果としてのガントチャートを示している。

これらのケースではいずれもロットの終了時間は同じとなっている。しかし表１ではプレスと塗装が連続した仕事割当になっているが、表２、３では隙間時間ができる。他方１個目の製品が一番早く完成するのは表３である。このように必要な材料の投入タイミングも製品の搬出のタイミングもディスパッチルールにより異なる。またより複雑なワークフローではロットの終了の時間も大きく変わってくる。このワークフローのロット（クラスター）単位でのスケジューリングについては、現在理論的に十分な解析は行われておらず、効果的な最適化のアルゴリズムも存在しない。しかし問題領域毎に適切なディスパッチルールを求めることは可能である。

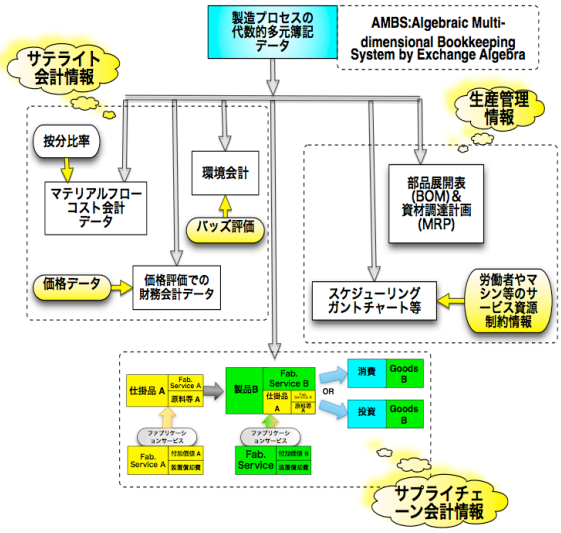


fig. 19 物作りに於けるデータフロー

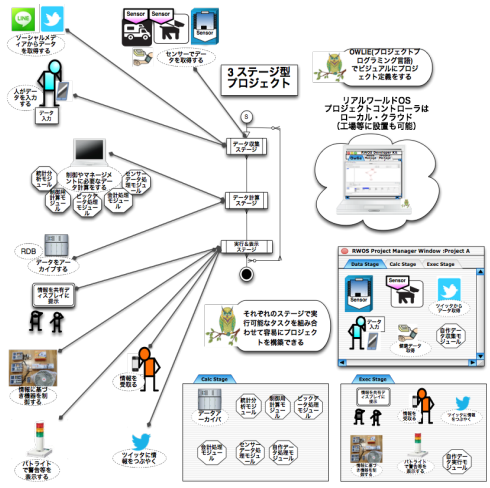


fig. 20 マイクロサービスのモジュールライブラリー

図１９は、タスクに対する代数的多元簿記の交換代数での特徴付けを上流としてマテリアルコスとフロー会計や通常の財務会計、環境会計、部品展開表とMRP、更にスケジューリングのガントチャートなどが計算に依って導出されることを示したものである。これにより組織間の製品のトレーサビリティやサプライチェーンBOMによる部品のトレーサビリティなども容易に実現可能となる。

【結語】

家庭や地域社会からビジネス・産業構造・一国経済・グローバル経済まで様々な人間活動システムが地球規模でのネットワークと共棲することが求められる時代に、その基盤となるアーキテクチャはオープンである必要がある。本稿ではそのためのオープンなアーキテクチャとしてリアルワールドOSとその階層的なアーキテクチャを示してきた。ここで我々が示したアーキテクチャは、個々のタスク遂行の現場での知を活かし、ボトムアップにマイクロサービスから全体のワークフローを構築することを可能にする枠組みとなっている。図２０は３ステージモデルで、マイクロサービスのライブラリーを用いて容易にワークフローによるサービス構築が可能となるためのライブラリーのビジョンを示したものである。様々なマイクロサービスのためのタスクのライブラリーが構築されオープン化され、更にワークフローそれ自体のライブラリー化が進む事で、第二次インターネット革命は、我々の経済社会システムの付加価値形成のありかたを根底から変えていく新たな流れを生み出していくことが期待される。

参考文献

[CISCO, 2013]シスコ IoT インキュベーションラボ, 『Internet of Everythingの衝撃』、インプレスR&D、2013

[Deguchi, 2004] H. Deguchi , Economics as an Agent Based Complex System, Springer-Verlag, 2004

[Deguchi, 2011] 出口弘, 市川学, 石塚康成, 志手一哉, 染谷俊介, 湯浅洋一,並列プロジェクト・タスク処理への多能工割付けの動的スケジューリング,国際P2M学会誌,国際P2M学会,2011　Oct,Vol.6, No.1, pp.179-189

[Deguchi, 2014A] 出口弘, POE(Point of Event)データとその利活用ーIOE時代に向けての個人・企業・政府のデータの利活用のための三つの原則、システム／制御／情報,VOL. 58,NO.7,PP.274-281,2014

[Deguchi, 2014B] 出口弘、サービスチェインと仕組みビジネス、情報処理、pp.140-147, Vol.55, No.2, Feb, 2014

[Deguchi, 2014C] 出口弘, 社会シミュレーションと組織・社会の情報処理のアーキテクチャ・デザイン, 情報処理, Vol.55, No.6, pp.539-548, 2014

[Deguchi, 2015A] 出口弘, IOE時代のP2M支援環境としての実世界OS - Real World OS as an Infrastructure of P2M for IOE Era -, 国際P2M学会誌, Vol.9 No.2, pp.99-121, 2015

[Deguchi, 2015B] 出口 弘, 竹林 知善, 吉田 宏章, 梅宮 茂良, 紺野 剛史, 石塚 康成, 木寺 重樹, 倉田 正, Shuang Chang, エネルギー会計によるエネルギー運用計画デザイン, 国際P2M学会誌, Vol.10 No.1, pp.191-214, 2015

[Deguchi, 2015C] 出口弘、IoE時代のもの・サービスの生産支援システム- 代数的多元簿記に基づく自律分散協調型システムとして -、経営情報学会２０１５年度秋季大会予稿集

[Mattessich, 2007] Richard Mattessich, Two Hundred Years of Accounting Research- Routledge New Works in Accounting History, Routledge, 2007

[pwc, 2014] technology forecast, Issue 1, 2014, プライスウォータハウスクーパース株式会社