Le formalisme EPO + CS : Un métamodèle conceptuel pour la modélisation des systèmes

Christophe Sibertin-Blanc
Institut de Recherche en Informatique de Toulouse, Université Toulouse
courriel : sibertin@irit.fr

Résumé : La modélisation est à l'évidence une activité centrale pour comprendre le monde et y intervenir, pour formuler des questions tout autant qu'apporter des réponses. Ces modèles prennent des formes très diverses, depuis les représentations mentales plus ou moins conscientes jusqu'aux théories mathématiques les plus abstraites, en passant par des formalismes dont la sémantique est définie de façon plus ou moins rigoureuse. Dans tous les cas, ces modèles incorporent un méta-modèle répondant implicitement à la question « qu'est-ce qu'un système ?». Cet article propose un cadre conceptuel pour apporter une réponse à cette question, motivé par l'idée que l'opérationnalité de toute modélisation gagne considérablement à expliciter le méta-modèle dans lequel elle s'inscrit.

mots clés : formalisation, modélisation

Pour produire un modèle d'un système, un analyste a besoin d'acquérir de la connaissance concrète sur ce système, quel que soit le projet qui sous-tende cette modélisation. Cette connaissance ne peut être acquise qu'auprès de personnes qui dispose d'une expérience pratique de ce système – qu'elles soient utilisatrices de fonctions offertes par le système vu comme un outil, qu'elles participent à son fonctionnement, à sa gestion ou son pilotage, ou encore qu'elles possèdent une expertise sur le domaine considéré.

Pour que ce transfert de connaissances puisse s'opérer dans de bonnes conditions et soit effectif, il faut que les spécialistes du système et l'analyste partagent une certaine représentation du monde. Il faut qu'ils disposent d'un référentiel commun, qu'il partagent un cadre conceptuel grâce auquel ils donnent le même sens aux concepts communément utiliser pour discuter à propos d'un système, tels que structure, acteur, flux, fonctionnement, événement, état, finalité, etc. C'est par exemple typiquement le cas du formalisme *entité-association* de la méthode Merise, utilisé pendant les phases amont d'un projet logiciel pour exprimer ce que l'on appelle le *schéma conceptuel* d'une base de données. Ce formalisme est suffisamment proche de la façon de voir les choses des utilisateurs ou des prescripteurs d'un système d'information pour qu'ils puissent valider la structure des données décrites par un schéma conceptuel, et il est tout à la fois suffisamment rigoureux pour que les informaticiens puissent, à partir d'un schéma conceptuel, élaborer le schéma physique de la base de donnée qui sera implantée sur ordinateur.

Si maintenant on considère toutes les dimensions d'un système, et non pas la seule structure de ses données permanentes, les choses deviennent beaucoup plus difficiles pour peu que le système auquel on s'intéresse soit assez important. Le modèle complet du système n'étant pas compréhensible dans sa globalité, on utilise des vues partielles du modèle du système, à la fois pour l'élaborer et pour en produire des représentations partageables. Ces vues, qui prennent sovent la forme de *diagrammes* dont la présentation graphique facilite l'intelligibilité, sont des projections du modèle qui ne prennent en compte que certaines dimensions du système (par exemple les données, les flux de communication ou bien les postes de travail), certaines de ses parties après l'avoir décomposé de façon hiérarchique en sous-systèmes plus ou moins indépendants, ou bien décrivent le système à un certain niveau d'abstraction ou de détail. Par exemple, la méthode UML propose une dizaine de diagrammes différents qui chacun peuvent être utilisés à des fins et dans des contextes différents. On se trouve alors très rapidement confronté à un grand nombre de diagrammes qui se recoupent en partie, dont les relations sont difficiles à comprendre parce que mal définies, et dont la

cohérence – qui veut que, pris tous ensembles, ils décrivent un système susceptible d'exister ou d'être réalisé – est difficile voire impossible à vérifier.

Pour échapper à ces difficultés, il faut que la relation entre le modèle du système dans sa totalité et chacun des diagrammes auxquels il peut donner lieu soit bien définie, que l'on puisse caractériser sans difficulté ce sur quoi porte un diagramme, interpréter facilement un diagramme dans les termes du système que l'on considère, ou encore comparer rapidement deux diagrammes. Cela suppose, selon nous, d'avoir la possibilité d'élaborer un modèle du système qui soit exprimé éventuellement en langage naturel mais en référence à un cadre conceptuel partagé par les personnes qui possèdent la connaissance pratique du système et par les analystes qui pousseront la modélisation jusqu'à son terme, de façon plus technique. Ce cadre conceptuel doit apporter une réponse tout à la fois raisonnablement simple et précise, non ambiguë, à la question *qu'est ce qu'un système ?*

L'objet de cet article est de présenter un tel cadre conceptuel. En résumé, un système comporte des *Entités*, concrètes ou abstraites, qu'il manipule, échange avec son environnement ou utilise comme ressources ; à chaque entité est associé un ensemble se *services* qui permettent de la créer, la stoker, la transformer ou de produire des information relatives à cette entité ; un système comporte aussi un ensemble de *Processeurs*, des unités de traitement ou acteurs qui ont les mêmes caractéristiques que les entités et sont de plus capables de conserver des entités et d'effectuer des O*pérations*, et pour ce faire consomment de l'énergie. Les concepts d'entité, de processeur et d'opération ont en commun de supporter la distinction *type / instance*, c'est à dire la concrétisation dans l'espace et le temps d'une forme abstraite (le type). Ces concepts ont un autre point commun : on peut leur appliquer le processus très générale de *raffinement*, qui permet de réaliser des descriptions qui se situent à des niveaux de détails différents sont liées récursivement par la relation quoi - comment.

L'ensemble des entités, processeurs et opérations d'un système constituent ce que l'on peut appeler sa structure, les éléments constitutifs qui déterminent ses fonctionnements possibles. Ces éléments sont mis en oeuvre dans des *Actions*, c'est à dire l'association d'un processeur (qui réalise l'action), d'entités (qui sont l'objet de l'action) et d'une opération qui est réalisée. Dans l'action « jean mange la pomme », on trouve bien un processeur, Jean, une entité, la pomme, et une opération, manger, qui est appliquée à l'entité par le processeur.

Le fonctionnement d'un système, qui lui permet entre autre d'interagir avec son environnement et qui justifie sa consommation d'énergie, est déterminé par sa *Structure de Contrôle* qui décrit les conditions de réalisation et les effets des actions.

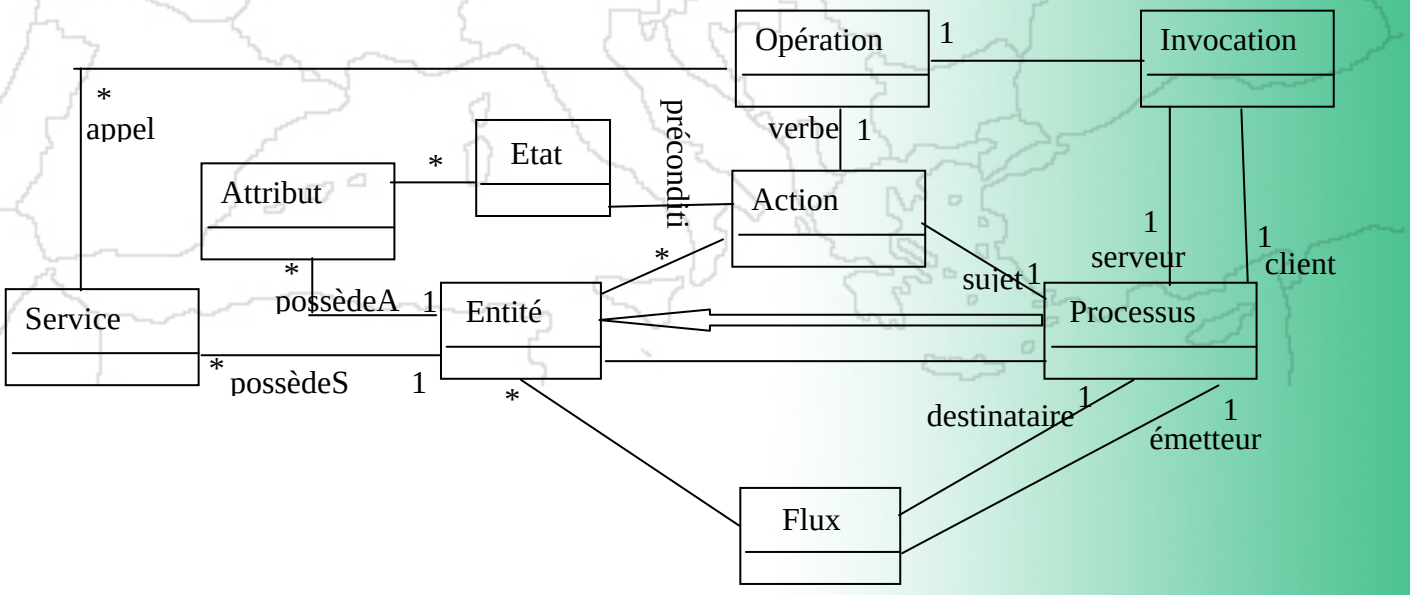

Figure 1 : les principaux concepts du méta-modèle

II. Entités, Processeurs et Opérations : la structure d'un système

Entités, processeurs et opérations sont les éléments constitutifs que l'on trouve dans tout système, dont la présence et l'assemblage sont indispensables à l'existence de ce système en tant que dispositif capable de faire preuve d'une certaine activité. Un système dépourvu d'entités ne saurait que broyer du vide, sans opérations aucune activité qu'il soit capable de faire n'est envisagée, et les processeurs sont indispensables à l'actualisation de cette activité potentielle. Nous allons introduire successivement ces trois sortes d'éléments, mais sans pouvoir le faire indépendamment l'une des autres : chacune ne prend son sens que dans ses relations avec les autres.

Les entités

Tout système manipule un certain nombre d'objets, de choses, d'éléments que nous appellerons des entités. Certaines de ces entités sont constitutives de l'existence du système : elles en font partie dès sa création et y perdurent jusqu'à sa disparition. D'autres entités ont une présence dans le système, ou même une existence, plus éphémère. Ce caractère passager ou temporaire de certaines entités tient à ce qu'un système peut échanger des entités avec son environnement, qu'il les reçoive en entrée ou qu'il les fournisse en sortie. Un système peut aussi créer de nouvelles entités, soit ex nihilo s'il s'agit d'entités abstraites ou informationnelles telles qu'une facture ou un nouvel élément d'une nomenclature, soit à partir d'autres entités s'il s'agit d'entités physiques constituées de matière, telle une `roue de vélo` assemblée à, partir d'une `jante`, de `rayons` et d'un `moyeu`. Symétriquement, un système peut faire disparaître certaines entités, qu'il les consomme, les détruise ou les décompose.

Les entités sont *discernables* les unes des autres, il est possible de distinguer une entité d'une autre, ne serait-ce, dans le cas des entités physiques, que par l'emplacement que chacune occupe à un instant donné ; si tel n'est pas le cas, sans doute convient-il de considérer l'ensemble de ces éléments indifférenciés comme étant une entité du système et non pas chacun d'eux individuellement ; si l'on s'intéresse au fonctionnement d'un bureau de poste, les propriétés de la `file d'attente` qui se trouve devant le guichet importent bien davantage que les caractéristiques de chacune des `personnes` qui composent cette file.

Une entité est caractérisée par un ensemble de propriétés, caractéristiques ou paramètres que nous appellerons ses *attributs* ; ces attributs, en nombre nécessairement fini, sont les informations pertinentes qu'il convient de considérer pour décrire (l'évolution de) l'état de cette entité, la façon dont elle peut être utilisée, transformée, déplacée, conservée, bref manipulée d'une façon ou d'une autre et contribuer au fonctionnement du système. A chaque attribut d'une entité est associée une valeur ; cette valeur est un élément du *domaine de valeur* de l'attribut, ensemble dont la délimitation est plus ou moins rigoureuse en fonction de la précision du modèle ; par exemple, le domaine de valeur d'un attribut `âge` pourra être l'ensemble des entiers de 0 à 200 tout aussi bien que l'ensemble des nombreuses expressions évoquant la durée d'une existence. Des entités figurent parfois parmi les caractéristiques que l'on a besoin d'attacher à une entité ; par exemple, la `commande` à laquelle se rapporte une `facture` dans un système commercial, les `père` et `mère` d'une `personne` dans un système ayant à voir avec l'état civil, ou la `pièce` se trouvant sur le `tour` d'une fraiseuse dans un atelier de production. Ces attributs dont le domaine de valeur est un ensemble d'entités sont communément appelés des *références*.

Si cela s'avère utile, il est possible d'associer à une entité un certain nombre de *services*, chaque service étant un traitement qui transforme l'entité d'une façon ou d'une autre. On trouvera notamment deux services standards, l'un pour créer l'entité et l'autre pour la détruire. Selon la typologie de [Lemoigne 99] l'effet d'un service sur une entité est de préserver son

existence dans le temps (par exemple enregistrer, archiver, sauvegarder, stoker, mémoriser, accéder, consulter, etc.), de la déplacer dans l'espace (par exemple communiquer, transférer, déplacer, envoyer, émettre, recevoir, récupérer, etc.), de modifier son état en changeant la valeur de l'un ou plusieurs de ses attributs, ou encore de la transformer en modifiant sa forme c'est à dire l'ensemble des attributs qui la caractérise.

On applique aux entités le principe de classification, qui consiste à faire abstraction du caractère unique de chaque entité pour identifier et caractériser ce que certaines d'entre elles ont en commun, à savoir leur structure. Le regroupement de l'ensemble de toutes les entités ayant exactement les mêmes attributs constitue une *classe d'entités*, et on dira que chaque entité est un élément, ou un *membre* de sa classe d'entité. Ce que les entités d'une même classe partagent en commun, c'est à dire la liste de leurs attributs et de leurs services, définit un *type d'entités*. Un type d'entité est donc comme un moule permettant de fabriquer de nouvelles entités et on dira que chaque entité est une occurrence, ou une *instance* de son type d'entité. En adoptant la terminologie de [Lemoigne 99], un type d'entité est une *forme* et une entité est une matérialisation de cette forme dans le *référentiel espace-temps*. (On peut noter cependant l'existence d'entités purement abstraites, telles que les instances du type `Maladie` : l'instance `rougeole` de ce type d'entité n'a aucune existence concrète qui permette de la localiser dans le temps et l'espace, seules peuvent être observées les affections de type `rougeole` dont souffrent certaines personnes à une période donnée ; les instances d'un type d'entité abstrait restent donc des formes hors de l'espace et du temps, mais enrichies par les informations spécifiques associées à chacun de leurs attributs). La classification est un procédé de modélisation extrêmement puissant : au lieu de décrire la structure de chaque entité une par une au moment de son introduction dans le système, on décrira une fois pour toutes les types des entités susceptibles de figurer dans le système, et l'introduction d'une nouvelle entité se limitera à signaler l'apparition d'une nouvelle instance d'un certain type, en précisant la valeur de ses attributs.

Dans la façon de décrire les entités que nous venons d'introduire, le lecteur familier du formalisme Entité-Association aura reconnu des entités dont on a internalisé les associations auxquelles elles participent, et celui familier avec les concepts de l'approche objet aura reconnu des objets privés de leurs méthodes. D'une façon plus générale, tout formalisme adéquat pour définir une structure informationnelle peut être utilisé pour décrire les entités d'un système (par exemple les procédés de construction des types d'un langage de programmation, un langage de définition de schémas de bases de données, ou encore un langage de représentation des connaissances). La façon de définir les entités présentée ici peut être enrichie en ayant recours à des mécanismes qui permettent de structurer plus rigoureusement l'ensemble des types d'entités d'un système, notamment des relations entre classes d'entités telles que *l'héritage* (si la classe B hérite de la classe A, une instance de B a les mêmes attributs qu'une instance de A, plus d'autres qui sont spécifiques aux instances de B), le *sous-typage* (la classe B est un sous-type de la classe A si dans tout système une instance de A peut être remplacée par une instance de B) ou bien la *composition* (une référence de la classe B vers la classe A est un lien de composition si toute instance de B est constituée, entre autres, d'une instance de A qui lui est indissociable). Un autre procédé permettant de renforcer la structure de l'ensemble des entités d'un système consiste à énoncer des *invariants*, ou contraintes d'intégrité, sous la forme de prédicats que les entités doivent satisfaire en toutes circonstances, par exemples : « les valeurs successives de la propriété âge d'une entité personne sont croissantes », « les deux parents (biologiques) d'une personne sont de sexes différents » ou encore la relation fonctionnelle qui relie le montant d'une facture avec la tarification appliquée aux éléments de la commande correspondante.

Les Processeurs

Si les entités sont les composants passifs d'un système, les processeurs en sont les composants actifs. Le système lui-même dans sa globalité est un processeur, et ses processeurs sont les différents sous-systèmes dont il est composé. L'environnement du

système est de même considéré comme un processeur, ou comme un ensemble de processeurs s'il est pertinent d'y distinguer différents secteurs. Tout ce que nous avons dit à propos des entités s'appliquent aux processeurs : certains processeurs sont constitutifs de l'existence du système alors que d'autres sont passagers ou éphémères (notons cependant que la plupart des systèmes sont composés de processeurs stables, la variation du nombre et de la nature des processeurs est une propriété que partagent les systèmes vivants, ou auto-poïétique, et les systèmes logiciels) ; les processeurs sont bien individualisés et discernables les uns des autres ; un certain nombre d'attributs déterminent les caractéristiques de chaque processeur ; les services associés à un processeur déterminent comment il peut être utilisé, modifié d'une façon ou d'une autre ; le principe de classification qui permet d'identifier des types de processeurs leur est applicable (on peut ainsi considérer le type `bureau de poste` que l'on distinguera de ses 18 000 instances ((à vérifier)) en France) ; l'ensembles des types de processeurs d'un système peut être structuré à l'aide de différentes relations telles que l'héritage, le sous-typage ou la composition ; enfin il est possible d'énoncer des invariants sur les processeurs d'un système, assertions qui permettent de raisonner sur ce qui peut advenir.

Les processeurs d'un système sont les organes qui *exécutent le travail* réalisé par ce système. Ce sont donc eux qui *hébergent les entités*, les stockent, les conservent, bref leur assurent l'existence, et leurs *appliquent des traitements*, les utilisent, font en sorte que ces entités servent à quelque chose. Vis à vis des opérations, ce sont les processeurs qui, tout au cours du fonctionnement et de l'évolution du système, *sélectionnent les opérations* à réaliser, les *réalisent* et prennent acte des conséquences de ces réalisations. Chaque processeur d'un système peut donc être vu comme un moteur, un dispositif qui réalise un certain travail, et à ce titre *consomme de l'énergie*. Déterminer comment un système dépense l'énergie qu'il consomme est un procédé d'identification des processeurs d'un système qui est aussi universel que le deuxième principe de la thermodynamique.

Chaque processeur interagit avec d'autres processeurs qui forment son *environnement*, ces processeurs étant soit d'autres processeurs faisant partie du système soit (dans) l'environnement extérieur du système. Ces interactions se font par l'intermédiaire des entités ou par celle des services. Les interactions par l'intermédiaire des entités donnent lieu à des *flux d'entités* : à l'occasion de la réalisation d'une opération ou d'un service, un processeur émetteur envoie une entité, ou bien un paquet d'entités, qui est ensuite reçu par un processeur destinataire. (On ne considère donc que des flux point à point, l'émetteur connaissant nommément et ayant la possibilité de communiquer directement avec le destinataire ; si l'émetteur ne connaît pas l'identité du destinataire et est uniquement en mesure de le désigner par certaines propriétés distinctives, on considèrera qu'il adresse le flux à un processeur de routage en charge de faire suivre l'entité au processeur destinataire final ; ce processeur de routage peut aussi dupliquer l'entité qu'il a reçue et l'adresser à plusieurs processeurs, on parle alors de diffusion). On peut associer à chaque flux certaines caractéristiques relatives à son débit (par exemple le nombre d'entités envoyées par unité de temps, la variation de ce débit dans le temps, etc.) ou relatives au délai d'acheminent ; notamment on dira qu'un flux est *synchrone* s'il est possible de négliger le délai entre l'envoi d'une entité et son enregistrement à la réception par le destinataire. L'autre forme d'interaction entre processeurs est *l'invocation de service* ; dans ce cas, le processeur à l'initiative de l'interaction, que l'on qualifie de *client*, active un autre processeur, le *serveur*, en lui demandant de réaliser les traitements associés à ce service. Cette invocation est effectuée par le client à l'occasion de la réalisation d'une opération ou d'un service et elle peut être accompagnée d'une entité, ou d'un paquet d'entités, qui est alors utilisé par le serveur pour réaliser le service demandé ; ces entités sont qualifiés de *paramètres* de l'invocation. Certains services sont définis de telle façon que chaque invocation donne lieu à un flux d'entité en *réponse*, qu'il s'agisse d'un résultat, d'un compte rendu d'exécution ou d'un simple accusé de réception. Une invocation d'un tel service est dite *synchrone* si le processeur client bloque son activité jusqu'à ce qu'il ait reçu la réponse de la part du serveur. Pour exemplifier les choses, considérons un distributeur de boissons ; lorsque l'utilisateur insère une pièce de monnaie dans l'appareil, il y

a un flux d'entité, que l'on peut considérer comme synchrone dans la mesure où aucune autre interaction ne peut avoir lieu temps que le distributeur n'a pas pris en compte l'introduction de cette pièce ; lorsque l'utilisateur appuie sur le bouton correspondant au type de boisson qu'il veut obtenir, il y a invocation d'un service qui retournera en résultat soit la boisson demandée soit l'affichage d'un message d'impossibilité de la délivrer ; que cette invocation soit synchrone ou non dépend uniquement de l'utilisateur : elle est synchrone si l'appui sur le bouton paralyse l'utilisateur au point de le rendre incapable de faire quoique ce soit temps qu'il n'a pas pris sa boisson ou lu le message d'avertissement. De son côté, le distributeur ne peut bien sûr qu'émettre des flux vers l'utilisateur : flux de monnaie rendue, flux d'information sur l'afficheur, et flux de la boisson fournie.

Les Opérations

L'activité réalisée par un système met en jeu des concepts beaucoup plus difficiles à cerner. Cela tient à ce que le concept de *travail* (tel qu'il est défini en physique) étant essentiellement immatériel et n'étant associé à aucune perception sensible, il ne donne lieu pour la plupart des modélisateurs à aucune représentation mentale spontanée. Nous proposons d'aborder cet aspect de la modélisation d'un système avec les trois concepts de procédure, d'opération et de service qui correspondent à des grains d'analyse différents.

Le concept de *procédure* est à associé au système dans son ensemble perçu comme un tout. Une procédure est une succession d'exécution d'opérations, comportant notamment des interactions entre le système et son environnement, qui sont toutes causalement liées et s'enchaînent pour contribuer à la réalisation d'un certain objectif. On distinguera les *procédures offertes* déclenchées à l'initiative de l'environnement du système, les *procédures requises* déclenchées à l'initiative du système, et les *procédures homéostatiques* qui sont cycliques et concernent la régulation interne du système. Une procédure offerte correspond à une fonction réalisée par le système, à un certain mode d'utilisation (ce que UML appelle un « cas d'utilisation »), à une mission que le système est en mesure d'accomplir, à un objectif que l'environnement cherche à réaliser par ces interactions avec le système. Si le système que l'on considère est un artefact, ces procédures ou *objectifs d'utilisation* sont précisément ce qui fait l'utilité de ce système et justifie son existence dans un certain contexte socio-technique. En contre-partie de ces procédures qu'il offre à son environnement, un système sollicite les procédures requises mises à sa disposition par son environnement et qui lui permettent d'acquérir les ressources (notamment en énergie et en information) dont il a besoin pour bien fonctionner. Si l'on conserve l'exemple simplissime d'un distributeur de boisson, il offre une seule procédure à ses utilisateurs, servir une boisson, il requiert de la part de son environnement des procédures telles que celles permettant son réapprovisionnement et le relevé des pièces de monnaie, et il comporte une procédure homéostatique lui permettant de réguler sa température interne.

Le concept de *service* est associé aux entités et processeurs. Un service correspond à un traitement atomique qui s'applique à une entité ou à un processeur. Soit il se comporte comme une fonction qui calcule et fournit de l'information relatif à cet élément, sans produire aucun effet direct sur lui, soit il s'agit d'un outil de manipulation de cet élément, qui le modifie d'une façon ou d'une autre tout en préservant sa cohérence telle qu'elle est spécifiée par les invariants de cet élément. L'atomicité d'un service correspond au fait qu'il n'y a pas lieu de le décomposer en unités de traitements plus fines. Nous avons mentionné plus haut les services de création et destruction d'une entité (ou d'un processeur) et évoqué la typologie de [Lemoigne 99]. Ce que nous avons dit sur la façon dont un processeur peut déclencher l'exécution d'un service d'un autre processeur peut être généralisé : un processeur peut également invoquer un service d'une entité. Mais un service n'est jamais exécuté isolément pour lui-même, il l'est toujours dans le contexte d'une opération.

Le troisième concept pertinent pour analyser l'activité d'un système est celui *d'opération*, qui est associé aux processeurs et se situe à un niveau intermédiaire entre les procédures et les services : une opération est une étape dans le déroulement d'une procédure, et les relations

entre opération et procédure sont définies par la notion de *structure de contrôle* introduite dans la section suivante ; le détail du traitement réalisé par une opération peut être décrit algorithmiquement en termes de services associés aux entités et processeurs mis en jeu par la réalisation de cette opération (il en résulte que l'identification des services n'est utile que si l'on a besoin de détailler ce que fait chaque opération). Le propre d'une opération est de préserver la cohérence du système : si une opération est appliquée alors que le système et toutes ses entités et processeurs sont dans un état cohérent et satisfont les invariants qui les concernent, alors il en est de même à l'issus de la réalisation de cette opération. De ce fait, nombre d'opérations s'appliquent à plusieurs entités simultanément, une modification de l'une entraînant automatiquement un changement dans une (ou plusieurs) autre, ou bien des informations dont l'une est porteuse étant nécessaires, comme un catalyseur, à l'exécution de l'opération. Par exemples, l'opération consistant à évacuer la pièce se trouvant sur le tour d'une fraiseuse tout à la fois déplace la pièce et libère le tour ; régler une facture crédite le compte du client et en même temps solde cette facture ; ou encore, le calcul du montant d'une facture ne peut être faire sans accéder aux tarifs qui doivent lui être appliqués.

La distinction type / instance (ou occurrence) que nous avons relevée à propos des entités et des processeurs s'applique pareillement aux opérations, procédures et services : le type d'une opération (et de même en ce qui concerne une procédure ou un service) est la description du travail à réaliser et il ne fait référence qu'à des types de processeurs et d'entités, alors que *l'occurrence d'une opération* est la réalisation effective de ce travail à un instant particulier et par des instances bien définies de processeurs et d'entités. Il est remarquable que le principe de classification qui conduit à cette distinction type / instance s'applique à chacune des trois sortes d'éléments du modèle d'un système, à savoir les entités, processeurs et opérations.

Un autre principe est applicable à chacune de ces trois sortes d'éléments, la *décomposition hiérarchique* d'un élément en éléments composants de même sorte que lui-même. En ce qui concerne les processeurs, est familière la décomposition d'une entreprise en départements ou directions, puis de chaque département en services, de chaque service en bureaux et finalement de chaque bureau en employés. En ce qui concerne les traitement, la pratique du *raffinement* est indissociable de la conception d'un algorithme, ou plus généralement d'une fonction un tant soit peu complexe réalisée par un système ; c'est le procédé qui permet d'identifier et d'organiser progressivement ce qui doit être fait, jusqu'à ce que l'on arrive à des éléments de traitements élémentaires qu'il n'est plus utile de détailler ; il ne s'agit là de rien d'autre que de la méthode d'analyse de Descartes ! Ce procédé ouvre la possibilité de réaliser des modèles d'un système qui se situent à des niveaux d'abstraction différents. On peut par exemple commencer par un premier modèle, le plus abstrait possible, ne comportant que le système lui-même pour seul processeur, ayant pour entités celles qu'il échange avec son environnement, et dont les opérations sont exactement les procédures qu'il offre à son environnement. Ce modèle initial peut être raffiné en un autre modèle plus détaillé dans lequel des sous-systèmes sont identifiés, avec les opérations qu'ils réalisent et les entités intervenant dans la réalisation de ces opérations. On peut à nouveau raffiner ce modèle, jusqu'à ce que l'on produise à un modèle se situant au niveau de détail permettant au modélisateur de résoudre le problème qui est à l'origine de l'élaboration de ces modèles. L'utilisation de ce procédé nécessite de veiller à *l'homogénéité* de chacun de ces modèles, propriété caractérisant le fait que les différents éléments se situent tous à un même niveau d'abstraction ; un modèle dans lequel des détails très spécifiques sont mis au même niveau que des principes structurant est très difficile à comprendre et le plus souvent inutilisable.

III. Actions et structure de contrôle : le fonctionnement d'un système

Les éléments introduits dans la section précédente décrivent une infrastructure, un ensemble de ressources suffisantes (et nécessaires) pour qu'une activité puisse se développer, mais ils ne définissent aucune activité. Par contre, ils permettent de définir la notion *d'état du système*, comme la valeur prise par tous les attributs des instances d'entités et de processeurs du système[1], ainsi que les éléments de base de toute activité, c'est à dire les *actions*. Ensuite, le fonctionnement du système n'est rien d'autre que la combinaison de ces actions, et c'est la *structure de contrôle* du système qui détermine comment ces actions sont composées.

Les actions

Une action est essentiellement un changement d'état du système, c'est à dire la modification de la valeur de certains attributs d'entités ou de processeurs. Cette modification est le fruit de l'exécution d'un ou plusieurs services, ou plus globalement le fruit de l'exécution d'une opération (rappelons qu'un service réalise un changement d'état significatif pour une entité ou un processeur considéré isolément, mais qu'un service est toujours exécuté dans le contexte d'une opération, qui est l'unité de traitement significative du point de l'ensemble du système). Et, conformément à leur rôle des processeurs dans un système, cette opération est déclenchée et réalisée par un processeur. Une action est donc un triplet qui fait intervenir :

- un processeur qui réalise l'action (le sujet),
- une opération, qui définit la nature de l'action réalisée (le verbe),
- un ensemble d'entités et de processeurs (les compléments), auxquels l'action est appliquée ou qui sont des ressources nécessaires à sa réalisation.

Dans l'action « Jean partage la pomme avec Paul », l'opération `partager` est réalisée par le processeur `Jean`, et elle met en jeu l'entité `pomme` et le processeur `Paul`. Par contre, dans une action telle que « Jean s'endort » l'ensemble des compléments est vide[2].

Si l'on s'accorde sur cette façon de définir le concept d'action, il apparaît clairement que chacune des trois sortes d'éléments de la structure d'un système est nécessaire, tout autant qu'ensembles elles suffisent à définir toutes les formes d'action réalisables. Chacun des éléments du modèle d'un système prend la totalité de son sens par les actions dans lesquelles il figure, et donc par les autres éléments auxquels il se trouve associé dans ces actions. Notons au passage que le critère d'homogénéité du modèle d'un système dont nous avons souligné l'importance provient essentiellement de l'association des éléments de la structure du modèle dans les actions : différents éléments ne peuvent figurer ensembles dans une même action que si ils se situent à un même niveau d'abstraction.

Puisque la distinction type / instance est applicable à chacune de leurs composantes, elle l'est de ce fait aux actions. On pourra donc considérer des *types d'action*, constitués d'un type de processeur, d'un type d'opération et d'un ensemble de types d'entités ou processeurs, donnant lieu à des exécutions qui sont des *occurrences d'action*.

Le principe de raffinement qui lui aussi s'applique à chacune de leurs composantes pourra de même être appliqué aux actions.

---

[1] Cette définition de l'état s'applique aux systèmes *discrets* pour lesquels l'exécution des opérations, ou plus précisément des services, est considérée comme instantanée, l'état du système n'étant pas définie pendant le temps d'exécution d'un service ; par exemple, considérons le `service` créditer d'une entité `cpte bancaire` : le solde du compte est défini avant et après l'exécution du crédit, mais pas pendant qu'il s'opère. Dans le cas des systèmes *continus*, l'état du système est défini y compris pendant que les opérations et services s'exécutent, et des variables leurs sont attachées pour caractériser le degré d'accomplissement de ces exécutions en cours.

[2] On ne tient pas compte ici des actions, difficiles à modéliser de façon formelle, dans lesquelles plusieurs acteurs se synchronisent pour réaliser ensemble une opération commune.

La structure de contrôle

Le comportement d'un système est déterminé par sa structure de contrôle, c'est à dire par un ensemble de règles qui indiquent comment les actions réalisables par le système peuvent s'enchaîner les unes les autres. Plus précisément, la structure de contrôle indique qu'elles sont les occurrences d'actions qui peuvent se produire à partir d'un d'état, ou de façon équivalentes quels sont les états à partir desquels une occurrence d'action peut se produire. Dans le premier cas on privilégie les états du systèmes en indiquant pour chacun d'eux ce qu'il est possible de faire à partir de cet état, dans le deuxième cas on privilégie les actions en indiquant dans quelles circonstances il est possible de les réaliser [Simon 1969].

Il existe de (très) nombreuses façons de définir la structure de contrôle d'un système, à l'aide de formalismes appropriés permettant une présentation de ces règles plus synthétique que leur simple énoncé. Le formalisme de référence est celui des *automates*, qui est utilisé tel quel mais aussi pour définir la sémantique de formalismes de plus hauts niveaux ou qui s'en présentent comme des extension. Par exemple, on utilisera volontiers les *réseaux de Petri* [Murata 1989, Sibertin 2000] si l'on a besoin de mettre en évidence la multiplicité des occurrences d'actions qui peuvent avoir lieu simultanément. Nous n'en dirons pas davantage sur ces formalismes dont une présentation, même superficielle, irait bien au-delà de l'objet de cet article.

Quelque soit le formalisme retenu, on ne saurait modéliser d'un seul tenant la structure de contrôle de l'ensemble d'un système, sauf dans le cas d'un modèle très abstrait ou, ce qui revient au même, dans le cas d'un système très simple. Il y a essentiellement deux façons d'organiser le modèle de la structure de contrôle d'un système : par *procédures* en mettant en évidence la dimension fonctionnelle du système, et par *processeurs* en mettant l'accent sur sa dimension organisationnelle.

Nous partons du principe que toute action réalisée dans un système est finalisée, et donc se produit dans le contexte d'une procédure. Si l'on décrit la structure de contrôle de chacune des procédures du système et comment ces procédures se synchronisent éventuellement les unes avec les autres, la totalité de l'activité du système aura donc été prise en compte. Une procédure offerte par le système se déroule depuis son déclenchement par l'environnement, par l'intermédiaire de la réception d'un flux d'entité ou d'une invocation d'opération, jusqu'à ce que le système fournisse à l'environnement ce qu'il lui a demandé par le déclenchement initial. Décrire la structure de contrôle de cette procédure, c'est déterminer toutes les successions d'opérations qui peuvent ou doivent s'exécuter depuis l'état dans lequel le système se trouve lorsqu'il reçoit le déclenchement jusqu'à l'état dans lequel ont été exécutées toutes les opérations susceptibles de contribuer à la bonne réalisation de la procédure. Les choses se passent symétriquement dans le cas des procédures requises, les rôles du système et de l'environnement étant inversées, à ceci près que l'on s'intéresse exclusivement aux opérations réalisées par le système et pas à celles relevant du fonctionnement interne de l'environnement. Quant aux procédures homéostatiques, elles comportent toujours un *état initial*, relativement stable, qu'elles ne quittent que sous l'effet d'un événement déclencheur particulier pour réaliser un nouveau cycle de régulation ; le déroulement d'une telle procédure décrit comment elle quitte cet état stable et les enchaînements d'opérations qui lui permettent d'y retourner.

Il nous reste à définir comment les différentes procédures d'un système se synchronisent les unes avec les autres, par leur déclenchement mutuel ou au cours de leur déroulement en introduisant les concepts d'événement et de cycle de vie d'une entité.

Un *événement* est une situation qui se produit dans le système et est significatif vis à vis de son comportement. On distinguera classiquement quatre types d'événement :

- La réception d'un flux d'entité par un processeur et plus particulièrement par le système en provenance de son environnement, par exemple l'arrivée d'un flux de type `commande client` ;
- L'invocation d'un service ou d'une opération, par exemple l'arrivée d'une `demande de boisson` ;

- La satisfaction d'une condition particulière résultant de l'exécution d'un service, condition portant sur la valeur d'attributs d'une entité ou d'un processeur, par exemple le passage de l'attribut `stock` d'une entité `produit` en-dessous de son `seuil d'alerte` (événement conditionnel) ;
- La survenue d'une date particulière, ou l'écoulement d'un délai (événement temporelle).

Comme nous l'avons vu, une procédure offerte est déclenchée par un événement de type flux ou invocation provenant de l'environnement du système. Quant aux procédures requises ou homéostatiques, elles sont déclenchées par un événement de l'un de ces quatre types, en excluant toutefois les événements de type flux ou invocation provenant de l'environnement.

L'autre mode de synchronisation possible entre procédures est moins important et nécessite de rentrer dans des considérations plus techniques. Il tient à ce que l'état d'une entité ne permet pas toujours de donner immédiatement suite à une invocation de service. Par exemple, le service `réserver` d'une entité `produit` ne peut être réalisé si le niveau de son `stock` est nul. On sera donc amené à associer à certains types d'entité une structure de contrôle, que l'on appelle son *cycle de vie*, qui détermine dans quelles conditions ses services peuvent être exécutés. Les différentes procédures d'un système qui s'exécutent simultanément, en parallèle, sont contraintes à respecter le cycle de vie de chacune des entités qu'elles manipulent, ce qui peut occasionner certaines synchronisations dans leurs déroulements respectifs.

Cette organisation de la structure de contrôle d'un système est essentiellement fonctionnelle en ce que, fondée sur les procédures, elle met l'accent sur les échanges entre le système et son environnement, et donc sur les relations entre les entrées et les sorties du système.

Venons en à la seconde façon d'organiser la structure de contrôle d'un système qui, fondée sur les processeurs, met davantage en évidence la structure du système. Le principe de base est cette fois que toute action est réalisée par un processeur et donc que, si l'on décrit l'activité de chacun des processeurs et comment ces activités se coordonnent, la totalité de l'activité du système aura été prise en compte. Sauf exception, l'activité d'un processeur est cyclique et donc se modélise essentiellement de la même façon qu'une procédure homéostatique. Quant à la façon dont les processeurs communiquent les uns avec les autres, nous l'avons déjà évoquée dans la section précédente.

IV. Considérations méthodologiques

Ce méta-modèle, en ce qu'il décrit la structure du résultat d'un processus de modélisation, ouvre de nombreuses perspectives sur les façons d'organiser un processus de modélisation que nous n'aborderons pas ici faute de place [Sibertin 2000].

Bibliographie

[Lemoigne 1999] LEMOIGNE J-L, La modélisation des systèmes complexes. Dunod, 1999, 1ère édition en 1990.

[Muller 2000] MULLER P-A, GAERTNER N. , *Modélisation objet avec UML*, 2ème édition (2000), Eyrolles, ISBN 2-212-09122-2.

[Murata 1989] MURATA T., Petri Nets: Properties, Analysis and Applications; In *Proc. of the IEEE*, vol 77, n°84, April 1989.

[OMG 2002] *OMG Unified Modeling Language Specification* UML V1.5, http://www.omg.org/technology/uml/index.htm, Juin 2002

[Sibertin 2000] SIBERTIN-BLANC C., CoOperative Objects: Principles, Use and Implementation. In *Petri Nets and Object Orientation*, G. Agha & F. De Cindio Eds., Lectures Notes in Computer Science vol 2001, Springer-Verlag, 2000.

[Sibertin 2000] SIBERTIN-BLANC C., Using Petri Nets and Objects: A Formal yet Expressive Approach.. Dans : *Software Specification Methods: An Overview Using a Case Study*. Henry Habrias, Marc Frappier (Eds.), Springer-Verlag, p. 259-278, octobre 2000.

[Simon 1969] SIMON H. A.. *The Sciences of Artificial*, Cambridge, MA, The MIT Press. Traduction Française *Les Sciences de l'artificiel*, Galimard-Folio, Paris 2004.

[Tardieu et Al 83]H. Tardieu, A. Rochfeld, S. Colleti. *La méthode Merise : Principes et outils*. Editions d'Organisation,1983.