Configuration Based Workflow Composition

Patrick Albert
ILOG
Email: palbert@ilog.fr

Laurent Henocque
LSIS
Email: laurent.henocque@lsis.org

Mathias Kleiner
ILOG & LSIS
Email: mathias.kleiner@lsis.org

***Abstract*— Automatic or assisted workflow composition is a field of intense research for applications to the world wide web or to business process modeling. Workflow composition is traditionally addressed in various ways, generally via theorem proving techniques.**

The originality of this research stems from the observation that building a composite workflow bears strong relationships with finite model search, and that some workflow languages can be defined as constrained object metamodels [1], [2]. This leads to consider the viability of applying configuration techniques to this problem. Our main contribution is to prove the feasibility of such an approach, with some advantages and drawbacks compared to logical based techniques.

We present a constrained object model for workflow composition, based upon a metamodel for workflows and ontologies for processes and data flows. Experimental results are listed for a working implementation that generates complex interleaving composite workflows involving transformations, synchronization and branching constructs.

I. Introduction

Automatic or assisted workflow composition is a field of intense research for applications to the world wide web or to business process modeling. Workflow composition is traditionally addressed in various ways, generally via theorem proving techniques.

The originality of this research stems from the observation that building a composite workflow bears strong relationships with finite model search, and that some workflow languages can be defined as constrained object metamodels [1], [2]. Our main contribution is to prove the viability of applying configuration techniques to workflow composition, as an alternative to logically founded approaches.

The plan of the article is as follows. The current section I is introductory, and briefly presents configuration in Subsection I-A, its application to workflow composition in Subsection I-B and related work in I-C. Section II presents the activity metamodel in the form of a constrained object program. Section III details how the composition problem is stated. Section IV details the configuration process. Two experimental use cases are presented in Section V a well as experimental results obtained using the ILOG JConfigurator program. Section VI concludes and present research perspectives.

A. Brief introduction to configuration

A configuration task consists in building (a simulation of) a *complex product* from *components* picked from a catalog of *types*. Neither the number nor the actual types of the required components are known beforehand. Components are subject to *relations*, and their types are subject to *inheritance*. *Constraints* (also called well-formedness rules) generically define all the valid products. A configurator expects as input a fragment of a target object structure, and expands it to a solution of the configuration problem, if any. This problem is semi-decidable in the general case.

A configuration program is well described using an object model (as illustrated by Figures 1, 2, 3), together with well-formedness rules or constraints. Technically solving the associated enumeration problem can be made using various formalisms or technical approaches: extensions of the CSP paradigm [3], [4], knowledge based approaches [5], terminological logics [6], logic programming (using forward or backward chaining, and non standard semantics) [7], object-oriented approaches [8], [5]. Our experiments were conducted using the object-oriented configurator Ilog JConfigurator [8].

An object-oriented configurator like Ilog JConfigurator represents its catalog of types using an object model involving classes, attributes and relations between these classes. Class relations are inheritance or associations. A configurator's user defines an object model of the knowledge field he wants to implement, together with well-formedness constraints. Indeed, an object model alone is not usually sufficient to fully denote knowledge domain semantics.

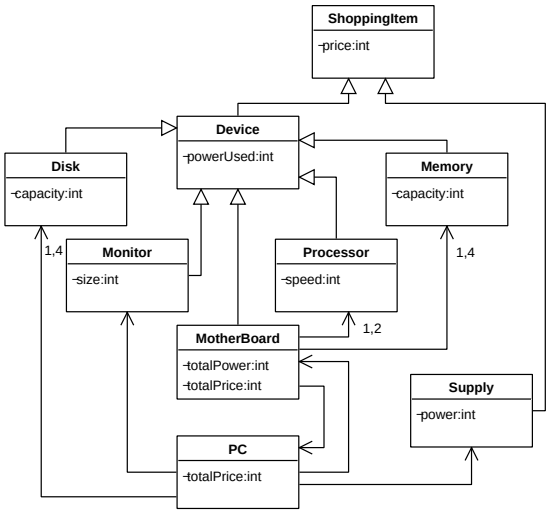

Fig. 1. A generic object model for PC configuration

For instance, to represent PCs[1], one may use the model

[1]I.e. Personal Computers

described in Figure 1. There we see that a *PC* must have exactly one *Motherboard*[2], exactly one *Power Supply*, and a unique *Monitor*. But it can have up to four *Disk(s)*[3]. The *Motherboard* can have one or two *Processor(s)* and one to four *Memory* units. Any (simplified) PC can be represented by an instance of this model. Additional constraints ensure that the instances are valid. For example, the attribute *totalPrice* of an instance of the class *PC* equals:

$$sum(MB.totalPrice, Su.price, Mon.price, Disks.price)$$

In this example, the dotted notation[4] is used to dereference the attributes of a class through associations, and $Disks.price$ represents the aggregated set of price values across all the disks known to the PC, no matter how many of them are present.

B. From configuration to workflow composition

Configuration emerges as an AI technique with applications in many different areas, where the problem can be formulated as the production of a finite instance of an object model subject to constraints. Reasoning about workflows falls into this category, because a workflow description is an instance of a given metamodel (as is the UML metamodel for activity diagrams [1]). Composing workflows is a configuration problem in that in so doing, one must introduce an arbitrary number of previously non existent transitions (fork, join, split, merge, transformations, pre-defined user-interactions sequences), and interconnect input and output message pins provided they have compatible types. The user of a composition system provides:

- a list of potentially usable workflows, implemented in the form of partially defined instances of the workflow metamodel used, (e.g. a producer for some good, a shipper, a payment service, etc.)
- the ontologies for the data types linked to the input/output messages present in the workflows (e.g. types of journeys: by train, plane, etc. , methods of payment: cheque, credit card, cash etc.)
- a goal to be satisfied by the result composition (e.g. a train ticket reservation),
- a list of the data input he can provide (e.g. simple "yes/no" answers, credit card number, expiry date, selected item etc.)

In our approach the goal is defined as the type in an appropriate ontology of a message connected to the final node of the composite workflow. The inputs provided by the user are modeled as external signals.

The user of a workflow composition system expects in return for his input a complete "*composite*" workflow, that interleaves the execution of several of the elementary argument workflows, while ensuring that all possible integrity constraints remain valid. Among such constraints are those that stem from the metamodel itself: for instance some constraints state that two or more workflows should not be inter-blocking, all waiting for some other to send a message. Other constraints are more problem specific, like those stating for instance that an item being shipped is indeed the one that was produced.

C. Related work

Automated workflow composition is a field of intense activity, with potential applications in at least two wide areas: Business Process Modeling and the (Semantic) Web Services. Tentative techniques to address this problem are experimented using many formalisms and techniques.

- Situation calculus [10]: In [11], the concurrent Golog extension ConGolog is shown suitable for Web Service composition. To circumvent the fact that the "sequence" Golog construct is static, and allows for no insertion of actions, they introduce an extraneous "Order" construct, that allows the dynamic insertion of an action so as to fulfill precondition conditions.
- Logic programming: the work in [12] illustrates the possibility of using Prolog to interactively generate Web Service compositions based on their semantic descriptions (originally in WSDL). This approach emphasizes the possibility of viewing Web Service composition as a recursive process, and advocates the use of well known AI techniques in the field.
- Type matching: [13] details an algorithm for Web Service composition with partial type matches, and shows that such an approach significantly improves the number of successful compositions. Interestingly enough, the composition algorithm interleaves the composition task with the Web Service discovery, which addresses several practical problems.
- Coloured Petri nets: [14] practically illustrates how coloured Petri nets can be generated from BPEL specifications, which allows detailed accounting for the compatibility of the choreographies. [15] details the viewpoints one can have when composing Petri net based choreographies and orchestrations, and the level of control that can be achieved. The intuitions in this article inspired us. We also reuse the producer/shipper example and compare our approach with the results given in [16].
- Linear logic: [17] proposes an application of LL to Web Service composition. The authors claim that the WSDL presentation of a Web Service can be automatically translated to a set of LL axioms. Then, they use a prover for the multiplicative propositional fragment of LL to infer the composition of a Web Service.
- Process solving methods [18]: PSM is not a formal system, but forms a model of processes that can be used to compose semantically described Web Services. The work in [19] describes a possible framework for using PSM in that objective. The intuitions underlying the PSM model can be related with practical experimentations conducted with the Ariadne mediator system, as documented in [20]. This work inspired ours to some extent.

[2]When missing, the cardinality defaults to one in our UML diagrams.

[3]Simple arrows with label X, Y represent a relation of cardinality X to Y

[4]This notation stems from the UML+OCL language [1] that has been identified as a rather appropriate language for specifying configuration problems in [9].

- AI Planning: In some aspects the problem of composing Web Services can be viewed as a planning problem. This particular view of the composition problem is covered by the work in [21], where state descriptions are ambiguous and operator definitions are incomplete. The same viewpoint is chosen in the library for interactive Web Service composition SWORD [22] where the plans are generated using a rule based forward chaining algorithm.
- Hierarchical Task Network (HTN) planning: an application of Shop2/BPEL to Web Service composition is presented in [23] and [24].
- Constraint programming: an original viewpoint over Web Service composition is advocated in [23] where the composition problem is viewed as a non-linear discrete optimization problem. The problem is called Activity Resource Assignment problem (ARA), in the presence of an optimization criterion.
- Markov decision processes: interleaving this with bayesian model learning, the algorithm in [25] generates workflows that are robust to non-determinism of Web services and that adapt to environment modifications.

II. A METAMODEL FOR WORKFLOW COMPOSITION

Workflow reasoning requires a workflow language with enough generality to be practically viable. Furthermore in our case, since we expect to treat workflow composition as a configuration task, it is of particular importance that the language is modular wrt. most if not all the workflow patterns referenced in [26]. The simplest such language is the extended workflow net YAWL language [2], now a subset of UML2 [1] activity diagrams.

We present our metamodel according to the standard model driven architecture recommendations. The next subsection introduces the classes, and their associations and attributes using class diagrams. This presentation is then followed by a detailed presentation of the OCL constraints. A global translation in the form of Ilog JConfigurator's object model and constraints is given last.

A. Metamodel class diagram

1) Activities: Activities are the core components of a workflow. As defined in UML2, activities may have a certain number of messages as their inputs and outputs. Those inputs/ouputs have defined types, taken from existing ontologies of data types. All activities have an "owner" (the original workflow they belong to : for instance, an activity may belong to the Producer workflow). In UML2, the owner of a workflow normally is a class in the object model, and related activities are usually graphically grouped together using partitions (the evolution for UML1.x swimlanes). All workflow parts that are dynamically added by the configurator in order to create the composed workflow belong to a newly introduced owner called the *Composition Workflow*. There are different types of activities, illustrated in the metamodel in Figure 2:

- Initial nodes: the starting point of the workflow. Initial nodes don't take any inputs. They are graphically represented using a black circle.
- Final nodes: a possible end of the workflow. Final nodes don't take any outputs and are represented using a white circle and a black dot in the center. There may be several final nodes in a workflow.
- Control nodes: joins, forks, merges, decisions. A fork initiates concurrency by duplicating its input token to all outputs. Join is the corresponding synchronization construct. Decision (also known as "split") and merge are the standard if/else conditional branching constructs.
- Actions: activities which include a local action executed by the workflow owner.
- Transformations: activities for transforming message data types with no further side effect. Transformations are called data mediators in the context of web service composition. Available transformations can be chosen by the composition designer, or they can be discovered (as e.g. in the context of Semantical Web Services).
- External Signals: An activity that outputs external messages, typically user provided messages.

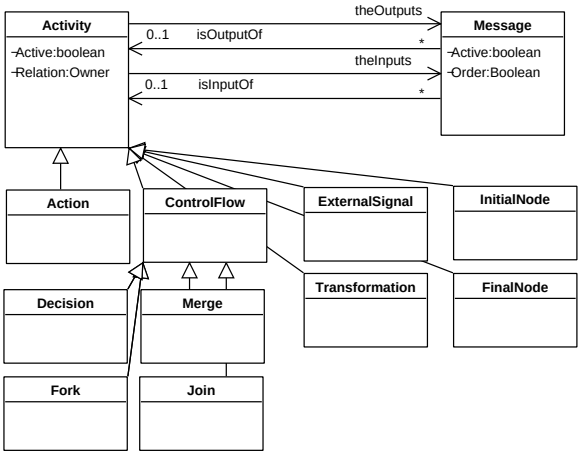

Fig. 2. Meta-model for workflows activities

2) Composition specific constraints: ¿From the simplified and slightly adapted subset of the UML2 activity diagram metamodel in Figure 2, we observe that both the *Activity* and *Message* classes implement a Boolean attribute called "active". This Boolean helps ensuring that a workflow can be composed from sub-workflows if and only if at least one valid path yields the expected goal. This allows our tool to produce composite workflows under the additional constraint that control flow constructs must match the following constraints applying to activities and messages:

- if an action is active, then all its input messages are active,
- if an active message is output of a join, then all inputs of the join must be active,
- if an active message is output of a decision ot a fork,

then the input of this activity must be active,
- if an active message is output of a merge, at least one of this merge's inputs must be active.

This allows the program to build solutions such that at least one path leads from the initial node to a final node reached via a message having the correct goal type. In the case a valid path traverses a fork or a join, all the other incoming/outgoing paths must be valid too. If a user wants a robust solution (meaning that all branches are valid), this can be obtained by forcing all parts of the workflow to be active.

3) Ontology of message types: Each message has a related data type, from a workflow specific ontology. We use predefined ontologies for user interaction schemes, and import the ones required by the selected web services. User interaction schemes, as implemented by the composition activity *Offer-Acceptance*, constrain the types of their I/O messages. From an abstract standpoint, they output an *OfferAnswer* if both an *Offer* and an *UserAcknowledgement* are provided. However the precise *Offer*/*OfferAnswer* type match is constrained: they must inherit from the same workflow datatype ontology. For example, a *ShipperOfferAnswer* can be output only if a *ShipperOffer* is input to the user interaction. Figure 3 illustrates the fact that such answers belong to both the hierarchy of standard datatypes and of imported service ontologies[5].

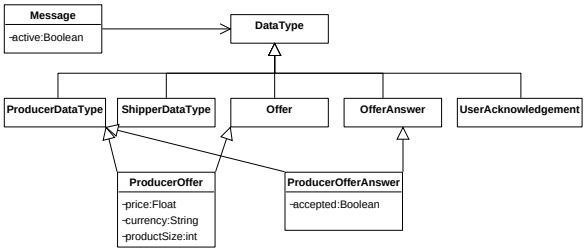

Fig. 3. Abstract model for workflows data types ontologies

B. Semantics

The previous object models are not enough to describe valid compositions, and require a number of constraints governing the possible combinations of partial workflows and additional elements that form acceptable compositions. We list here a limited number of these constraints that are representative enough to grasp the general idea, using the OCL language from UML2:

C. General constraints

These constraints are not problem-specific and therefore apply to any composition.

1) If an action is active, then all of its inputs must be active messages:
$Context\ Action$
$inv{:}self.active{=}true\ implies$
$self.inputs{\rightarrow}forAll(m{:}Message|m.active{=}true)$

2) If a merge is active, then at least one of its inputs must be an active message:
$Context\ Merge$
$inv{:}self.active{=}true\ implies$
$self.inputs{\rightarrow}exists(m{:}Message|m.active{=}true)$

3) All messages input of an external workflow are output of the composition workflow:
$Context\ Message$
$inv{:}self.isInputOf.Owner{<>}Composition\ implies$
$self.isOutputOf.Owner{=}Composition$

4) All messages output of an external workflow are input of the composition workflow:
$Context\ Message$
$inv{:}self.isOutputOf.Owner{<>}Composition\ implies$
$self.isInputOf.Owner{=}Composition$

5) The order of an activity's input message is lower than the activity's outputs orders (this "ordering" constraint allows to prevent loop situations in the composition workflow):
$Context\ Message$
$inv{:}self.outputOf.Outputs{\rightarrow}forAll(m{:}message|self.Order{<}m.Order))$

D. Pre-defined composition constraints

1) Both the class of an OfferAcceptance Action output and that of its Offer input are subtypes of the same service class (this constraint is required because the concept of an OfferAcceptance is generic).

2) Fork nodes have the same type for their inputs and outputs:
$Context\ CompoCopy$
$inv{:}self.outputs{\rightarrow}forAll(m.OclIsTypeOf(self.input))$

E. Problem specific constraints

1) User provided message types:
$Context\ ExternalSignal$
$inv{:}self.outputs{\rightarrow}forAll(\ m.theData.OclIsTypeOf(type1)$
$OR\ m.theData.OclIsTypeOf(type2))$

2) Available transformations: The problem input may list a number of useful transformations in that case. In the context of (semantic) web service discovery, such transformations may be the result of a query to a repository of ontology mediators.

3) Policy related constraints: Some constraints may be required to filter out valid yet unwanted compositions, for instance in a way such that offer's prices fall below a given maximum value .

F. Preferences

Finally, preferences may be stated among the solutions, which turn the problem into an optimization one. For example,

[5]It should be noted that JConfigurator does not support multiple inheritance, which leads to a slightly different implementation.

one may prefer solutions involving a possibility of a credit card payment, or where the number of user interactions is the least possible.

G. From UML to JConfigurator

We give here the translation of two of the previous constraints in the Java API provided with the JConfigurator constraint library.

- If an Activity is "Active", at least one input of the Activity is "Active":

$om.add(om.forAll(A,$
$om.eq(A.getIntField("Active"),1),$
$om.ge(om.sum(A.getObjectSetField("inputs"),"Active"),1)$
$));$

- For all activities, the max of inputs order is strictly less than the min of outputs order:

$om.add(om.forAll(A,om.lt($
$om.max(A.getObjectSetField("outputs"),"Order"),$
$om.min(A.getObjectSetField("inputs"),"Order")$
$)));$

III. The composition problem

A workflow composition problem (M_c, W, T, I_t, O_m) can be paraphrased as: "Given a constrained object metamodel M_c, a set W of partial workflows provided as candidates to composition, a set T of available transformations, a set I_t of possible input message types, and a *goal* in the form of a set O_m of typed output messages expected from the composition, produce a composite workflow involving a selection $W' \subseteq W$ of the candidate partial workflows, plus the required number of auxiliary workflow constructs including elements from a a subset $T' \subseteq T$ such that: a/ the composite workflow is *valid* and b/ it produces all the expected output messages O_m from a set of input messages taking their types from a subset $I'_t \subseteq I_t$ of the input types".

By "valid", we understand that the resulting composition workflow satisfies all the metamodel constraints (notably constraints that prevent inter-dependency leading to process starvation). Furthermore, we require that all the workflows participating in the solution (including the "user" which inputs messages), communicate their messages back and forth to the composition workflow. This last point may seem artificially complex since solutions to some composition problems (as the one in [19]) can be obtained by direct connection between workflows, but is required in more complex problems where constraints apply to the messages that are exchanged between connected participants (like in [16]), or more generally when the composite workflow involve preexisting components, that cannot establish direct connections to their peers (as e.g. web services).

A. Workflow instances

Each potentially participating workflow is defined as a partial instance of the workflow metamodel, involving activity nodes, typed input and output pins, and fork/join transitions. All the messages are typed. A central part in the configuration algorithm used relates with the connection of inputs to outputs or fork/join transitions. This model, which integrates both choreography[6] and orchestration[7] offers the advantage of allowing to reason about the composition integrity. Specially, since the relations between inputs and outputs in transitions imply temporal sequencing constraints, no composition can be produced that violates these constraints. As a result, the workflows that lead to process starvation cannot be generated.

B. Composition transitions and transformations

Our approach is currently limited to the generation of "simple" workflows, not involving loop operators. This however suffices to solve a rich variety of composition problems. One key issue in that respect is the availability of transformations, that act as data adapters within the data flow. Such transformations, called ontology mediators in the field of the semantic web, are absolutely necessary, because independent processes or web services may define compatible, yet non strictly matching interfaces.

Constraints bind the input and output types of transformations so that only abstract transformation instances need to be created as part of the configuration problem before the search begins.

C. The configuration request

We wish to solve the workflow composition problem (M_c, W, T, I_t, O_m) using configuration techniques. This can be viewed as the process of connecting the goal O_m to some user inputs taking their types from I_t, via a number of intermediate workflows, control flows, and transformations.

The choice of a specific configurator (here ILOG JConfigator) clearly influences the implementation of the composition problem into a composition request. One such "specificity" of JConfigurator is the absence of "creation by necessity"[8]. In the process of configuring, when the min cardinality of a relation port cannot be fulfilled by existing objects, some constraint based configurators may trigger the automatic creation of required instances of the correct type. This is so in the previous C++ version of JConfigurator but was dropped in the Java version. To circumvent the lack of creation by necessity, all the object instances that may participate to a solution must be provided *in sufficient number*. Otherwise, if objects are required but do not preexist in the domain, the configuration will fail or miss solutions. The difficulties incurred by this limitation however are limited because JConfigurator implements

[6]Choreography is a specification of the inputs/outputs together with a partial workflow that describes the externally visible transitions of a reference workflow

[7]Orchestration is a fully documented workflow that describes all its transitions.

[8]According to various viewpoints, this may or not not be seen as a limitation.

classification reasoning[9]: object instances may be created for abstract types, to be further classified during the search. The creation of a pool of abstract objects thus prevents having to prepare for instances of all the possible leaf types in a hierarchy.

Among the objects that must be created before search begins is the “composition” workflow owner (all activities belong to such a workflow “owner”). It prepares for interaction with all the workflows that may potentially enter the composition by holding a sufficient number of control flow transitions (fork,join,split,merge), as well as transformations and partial workflows that implement predefined user interaction schemes. Elements from these groups will be selected by the configuration process to participate to the composition.

We must formulate the composition request by providing the following elements:

- the composition workflow owner,
- one initial workflow node (that models the point where the workflow starts),
- several final workflow nodes (potentially modeling multiple workflow terminations: in simple situations, one is enough),
- the configuration goal is the input data type(s) for the final node(s), formulated as a classification constraint on each final node input message,
- all potential user inputs (e.g. say “yes”, give a credit card number, etc...) are given as possible external signal’s output types (recall that a signal is an activity of a special kind),
- a number of predefined acknowledgement transitions: recall that each time a message is sent to the user, he must reply. Such instances are generic, under the constraint that their input/output data types are compatible,
- a number of fork/join/split/merge transitions needed to synchronize between the outbound and inbound flows of the composition. Using these constructs, the configurator is able to generate solutions that send acknowledgement messages to the user, and to synchronize the user outputs.

IV. The configuration process

We are using a recursive object based configuration procedure. Configuration is *goal oriented*: it starts from the final node, and attempts to connect its input message to the output of either a partial workflow, or a transformation. Once done, this results in further connection requirements, potentially solved by connection to the user inputs, or to more workflow outputs or transformations. This approach is hence goal oriented and apparently mimics backward chaining techniques. In fact, the recursive composition process may follow various paths depending of the chosen heuristics. One important option in that respect is to let the search recursively traverse newly connected components immediately or to postpone this decision.

[9]JConfigurator dynamically refines the types of objects to match typing constraints incurred during search.

The constrained object model we are using however does not preclude other solving strategies, including more ”forward chaining like” techniques that would start from the inputs, or mixing both. It should be noted however that although the goal must be connected, some of the possible inputs may remain unused. For instance, only one payment method among several is used in an online transaction. This situation normally disqualifies user input driven (forward) approaches. Among the possibilities left open by the current framework is local search, a potential ”must have” to address scalability issues.

V. Experimental results

We relate here results obtained using our techniques on two problems obtained from the literature. These problems where chosen because of the range of difficulties that they involve. The experiments where conducted on 2.8 Mhz Mobile Pentium, having 1 GO of Ram, under Windows, and using the Java JVM 1.5 and JConfigurator 2.1. Computation times are in seconds. Choice point counts indicate the number of choice points that where opened to reach a solution, and fail counts indicate the number of backtracks.

A. Theater and Film

This problem stems from [19], that deals about the composition of semantic web services. This example presents the automatic execution of the following tasks: obtaining a theater name, then a timetable for a specific movie in this theater, and finally buying the ticket for the selected theater/film/schedule triple. We provide the same input details as in [19]. The resulting composition workflow is presented in Figure 4.

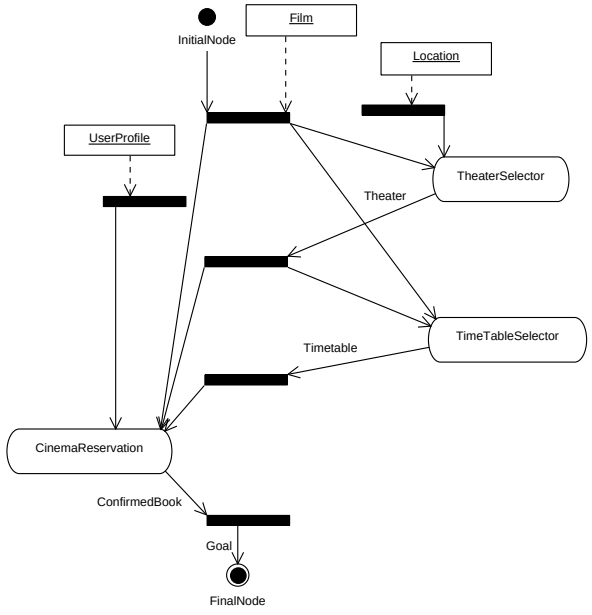

Fig. 4. The theater and film composed workflow

In Figure 4, we observe that all user inputs are directed to a specific join construct. This is so because as explained

before, all messages are exchanged through the composition workflow, which can redirect and/or copy the user provided messages to web services needing them. The solution being listed here does not require transformations either, although the problem data can be changed easily to involve this. We have mentioned a simple solution for clarity.

Note that the fork constructs are needed to copy data tokens to various places in the composition. This issue was kept implicit in [19] but requires a rigorous handling.

B. Producer and Shipper

This second problem originates from [16], an article dealing with web service composition using planners. Their Producer/Shipper example has been chosen because of its interesting properties:

- both the shipper and the producer make an offer corresponding to the user request, which are aggregated to make a global offer which can be accepted or rejected by the user. Note that the shipper needs input data from the producer to build its offer,
- both the producer and the shipper are specified using partial workflows, and do not simply amount to simple isolated activities,
- the two partial workflows cannot be executed one after the other, but they must be interleaved, as each one must wait for the other offer to obtain an OfferAcceptance and therefore complete the transaction,
- the ShipperWorkflow needs a size as input, which can only be obtained by extraction (i.e transformation) on the ProducerOffer,
- finally, the goal is decomposed into two sub-goals: the producer and the shipper order confirmations.

Figure 5 illustrates the shipper's partial workflow, as defined before search begins. The producer's workflow is similar, modulo the message type ontologies.

The composition result is shown in Figure 6. This composed workflow involves synchronization, interleaving, transformations (we used oval boxes to denote transformations) and it should be noted that some execution paths are discarded: indeed, under user rejection, the goal cannot be fulfilled. This illustrates why we needed to implement a Boolean attribute for active paths in the metamodel.

C. Execution statistics

We have conducted experiments on the two previous detailed examples. The resulting program statistics are listed in Table I. No heuristics have been used for computing these solutions, the system being run in default mode. As the results indicate, the important ratio of fails per choice points shows that the program failed to discover a solution quickly. This implies that appropriate heuristics should greatly decrease the computation time required to reach a solution, a current field of activity. It can be noted that in [16], the user workflow is given as input to the program (this is not the case here), and the time for computing the workflow is 18000 seconds[10]

TABLE I

EXPERIMENTAL RESULTS FOR THE TWO EXAMPLE PROBLEMS (TIMES ARE IN SECONDS)

Problem	time	choice points	fails
Theater	0.46	13	0
Producer-Shipper	20.06	103151	103109

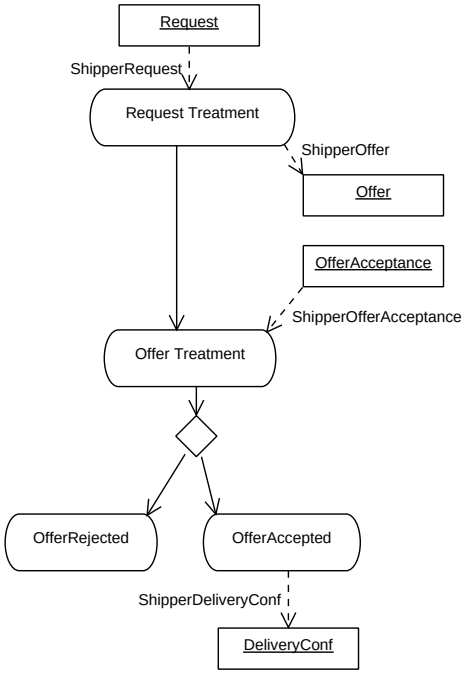

Fig. 5. The shipper provided workflow

VI. CONCLUSION

This research proves the feasibility of using configuration techniques to achieve automatic workflow composition, with possible applications to many areas, for instance web service or business process composition. This possibility acknowledges the fact that workflow composition can be seen as a finite model search problem.

Configuration expects a constrained object model to operate, hence place the application design in a field familiar to many engineers. An essential part of the object model, the metamodel for activity diagrams, already exists as a (subset of) part of the UML2 specification relative to activity diagrams.

Configuration also places the problem in the constraint programming paradigm. In so doing, policy based composition for instance is naturally (if not easily) dealt with as an optimization problem (finding a composition which minimizes a given criterion) or as preferential search problem driven by heuristics. These issues are relevant to policy based goal driven composition.

Being combinatorial in nature (a workflow is a graph), the problem we address is difficult. Configuration techniques may

[10]Experimental conditions differ significantly in both cases however, and computation times cannot be compared accurately.

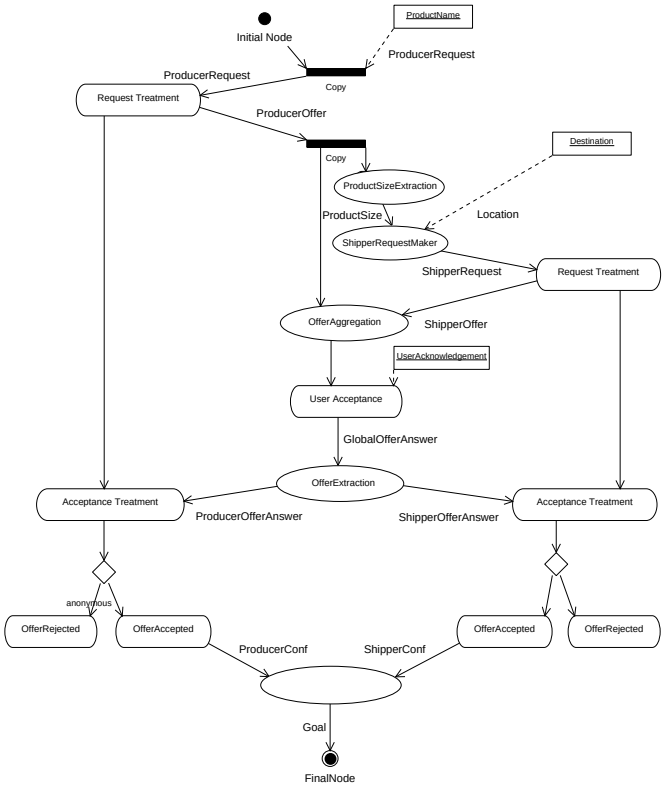

Fig. 6. The shipper and producer composed workflow

help tackle the complexity by:

- using abstractions, thanks to classification reasoning,
- using optimal data structures (since portions of the input partial workflows are invariant, they could be implemented using lighter hoc data structures)
- using local search techniques,
- using isomorphism rejection techniques, ...

All these options pertain to future or active research. We are currently working on scalability, increasing the number of irrelevant input workflows, not needed to compute a solution.

ACKNOWLEDGMENTS

The authors would like to thank the European DIP integrated project and the ILOG company for their financial support in carrying out this research.

REFERENCES

[1] O. M. Group, *UML v. 2.0 specification*. OMG, 2003.
[2] W. van der Aalst, L. Aldred, and M. Dumas, "Design and implementation of the yawl system. qut technical report, fit-tr-2003-07," Queensland University of Technology, Brisbane, Tech. Rep., 2003.
[3] S. Mittal and B. Falkenhainer, "Dynamic constraint satisfaction problems," in *Proceedings of AAAI-90*, 1990, pp. 25–32.
[4] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner, "Configuring large-scale systems with generative constraint satisfaction," *IEEE Intelligent Systems - Special issue on Configuration*, vol. 13(7), 1998.
[5] M. Stumptner, "An overview of knowledge-based configuration," *AI Communications*, vol. 10(2), pp. 111–125, June 1997.
[6] B. Nebel, "Reasoning and revision in hybrid representation systems," *Lecture Notes in Artificial Intelligence*, vol. 422, 1990.
[7] T. Soininen, I. Niemelõ, J. Tiihonen, and R. Sulonen, "Unified configuration knowledge representation using weight constraint rules," in *ECAI 2000 Configuration Workshop*, 2000.
[8] D. Mailharro, "A classification and constraint based framework for configuration," *AI-EDAM : Special issue on Configuration*, vol. 12(4), pp. 383 – 397, 1998.
[9] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "Configuration knowledge representation using uml/ocl," in *Proceedings of the 5th International Conference on The Unified Modeling Language*. Springer-Verlag, 2002, pp. 49–62.
[10] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
[11] S. McIlraith and T. Son, "Adapting golog for composition of semantic web services," in *proceedings of Conference on Knowledge Representation and Reasoning*, April 2002.
[12] E. Sirin, J. Hendler, and B. Parsia, "Semi automatic composition of web services using semantic descriptions," in *proceedings of the ICEIS-2003 Workshop on Web Services: Modeling, Architecture and Infrastructure*, Angers, France, April 2003.
[13] I. Constantinescu, B. Faltings, and W. Binder, "Large scale, type-compatible service composition," in *proceedings of IEEE International Conference on Web Services (ICWS 2004)*, San Diego, USA, 2004.
[14] X. Yi and K. Kochut, "A cp-nets-based design and verification framework for web services composition," in *proceedings of 2004 IEEE International Conference on Web Services, July 2004*, San Diego, California, USA, July 6-9 2004.
[15] R. Dijkman and M. Dumas, "Service-oriented design: A multi-viewpoint approach, ctit technical report series no. 04-09," Centre for Telematics and Information Technology, University of Twente, The Netherlands, Tech. Rep., February 2004.
[16] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso, "Planning and monitoring web service composition," in *proceedings of the Workshop on Planning and Scheduling for Web and Grid Services held in conjunction with ICAPS 2004*, Whistler, British Columbia, Canada, June 3-7 2004.
[17] J. Rao, P. Kungas, and M. Matskin, "Logic-based web service composition: from service description to process model," in *proceedings of the 2004 IEEE International Conference on Web Services, ICWS 2004*, San Diego, California, USA, July 6-9 2004.
[18] V. Benjamins and D. Fensel, *Special Issue on Problem-Solving Methods. International Journal of Human-Computer Studies (IJHCS)*, vol. 49(4), pp. 305–313, 1998.
[19] A. Gómez-Pérez, R. González-Cabero, and M. Lamaa, "A framework for design and composition of semantic web services," in *Semantic Web Services, 2004 AAAI Spring Symposium Series*, 22nd-24th March 2004.
[20] S. Thakkar, C. Knoblock, J. Ambite, and C. Shahabi, "Dynamically composing web services from on-line sources," in *proceedings of AAAI-02 Workshop on Intelligent Service Integration*, Edmondon, Canada, July 2002.
[21] M. Carman, L. Serafini, and P. Traverso, "Web service composition as planning," in *proceedings of ICAPS03 International Conference on Automated Planning and Scheduling*, Trento, Italy, June 9-13 2003.
[22] S. Ponnekanti and A. Fox, "Sword: A developer toolkit for web service composition," in *proceedings of the 11th International WWW Conference*, Hawaii, May 7-12 2002, p. to appear.
[23] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2," *Journal of Web Semantics*, vol. 1, no. 4, pp. 377–396, 2004. [Online]. Available: http://www.mindswap.org/papers/SHOP-JWS.pdf
[24] M. Vukovic and P. Robinson, "Adaptive, planning based, web service composition for context awareness," in *proceedings of the Second International Conference on Pervasive Computing*, Vienna, Austria, 2004, p. to appear.
[25] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic workflow composition using markov decision processes," *International Journal of Web Services Research*, vol. 2(1), pp. 1–17, Jan-March 2005.
[26] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, pp. 5–51, July 2003.