The Benefits of Rapid Modelling for E-business System Development

Juan C. Augusto, Carla Ferreira, Andy M. Gravell, Michael A. Leuschel, and Karen M.Y. Ng

Department of Electronics and Computer Science
University of Southampton,
{jca,cf,amg,mal,myn00r}@ecs.soton.ac.uk

Abstract. There are considerable difficulties modelling new business processes. One approach is to adapt existing models, but this leads to the difficult problem of maintaining consistency between model and code. This work reports an investigation into creating quick models that are nonetheless useful in providing insight into proposed designs.

1 Introduction

There are considerable difficulties modelling new business processes. One approach is to adapt existing models, but this leads to the difficult problem of maintaining consistency between model and code. In eXtreme Programming [Bec00], for example, we are advised to “travel light” – questions are answered by examining the code rather than trusting written designs which may be out of date.

This work reports an investigation into creating quick “throw away” models that are nonetheless useful in providing insight into proposed designs. These models are not merely pictures, but can be “executed” through animation/simulation, and can be comprehensively checked, at least for specific configurations, by model checking. This answers the criticism that pictures “can’t give concrete feedback” [Beck 2000].

In sections 2 and 3 we first provide a brief description of two of the modelling frameworks we considered, Promela/SPIN and CSP(LP)/CIA. Then in section 4 a case study that we used as the basis for our experiment is introduced. In section 5 we give some details about the modelling approach we followed when using Promela and CSP(LP) as modelling languages. Section 6 explains how tool assisted development can provide the basis for rapid modelling with important expected benefits and section 7 explains the extent to which we experienced those advantages while using the above mentioned tools to our case study. Later, in section 8 we consider a more specific modelling language, called StAC for business modelling. An analysis of work in progress is given in section 9 while an account of some lessons learnt and the final conclusions are provided in section 10.

2 Promela/SPIN

A model-checker

SPIN [Hol97] has been a particularly successful tool that has been widely adopted to perform automatic verification of software specifications. SPIN offers the possibility to perform simulations and verifications. Through these two modalities the verifier can detect absence of deadlocks and unexecutable code, to check correctness of system invariants, to find non-progress executions cycles and to verify correctness properties expressed in propositional linear temporal logic formulae. Promela is the specification language of SPIN. It is a C-like language enriched with a set of primitives allowing the creation and synchronization of processes, including the possibility to use both synchronous and asynchronous communication channels. We refer the reader to the extensive literature about the subject as well as the documentation of the system at Bell Labs web site for more details: `http://netlib.bell-labs.com/netlib/spin/whatispin.html` We assume some degree of familiarity with this framework from now on.

3 CSP(LP)/CIA

CSP(LP) [Leu01] unifies CSP [Hoa85] with concurrent (constraint) logic programming. Elementary CSP, without datatypes, functions, or other advanced operators, was extended in CSP-FDR [Ros99] to incorporate this features, which we want for modelling business systems. Some of the remaining limitations on pattern matching were overcome in CSP(LP) (see [Leu01] section 2.2. for a more detailed account). A basic introduction to CSP(LP) syntax [Leu01] follows:

Operator	Syntax	Ascii Syntax
stop	$STOP$	`STOP`
skip	$SKIP$	`SKIP`
prefix	$a \to P$	`a->P`
conditional prefix	$a?x : x > 1 \to P$	`a?x:x>1->P`
external choice	$P \Box Q$	`P [] Q`
internal choice	$P \sqcap Q$	`P |~| Q`
interleaving	$P \vert\vert\vert Q$	`P ||| Q`
parallel composition	$P\ [\vert A \vert]\ Q$	`P [| A |] Q`
sequential composition	$P ; Q$	`P ->> Q`
hiding	$P \setminus A$	`P \\ A`
renaming	$P[R]$	`P [[ R ]]`
timeout	$P \rhd Q$	`P [> Q`
interrupt	$P \triangle_i Q$	`P / Q`
if then else	$if\ t\ then\ P\ else\ Q$	`if T then P else Q`
let expressions	$let\ v = e\ in\ P$	`let V=E in P`
agent definition	$A = P$	`A = P;`

The CSP(LP) Interpreter and Animator, CIA, [LAB+01] can be used to animate and detect deadlocks in a CSP(LP) specification.

Only Animate and deadlock-check???

4 The Travel Agency Case Study

As an example of an e-business system involving collaboration across organizations consider a travel agent [LR00]. A travel agent gets requests from the users that log into the travel agency system by using a browser. After selecting an operation (book or unbook) and a service (car or room), the operation is submitted to the travel agent. The travel agent will decide what service provider (Car Rental or Hotel) to contact on the basis of previous requests made by the user. The request is passed on to one of the service providers, which will decide if the operation can be accomplished or not. For example, it could be that the user requests to book a service that is not available or to unbook a service that was not previously booked to her/him. The shop will contact the travel agent to make explicit if the operation was or was not successful. The travel agent will pass this information to the user. If the operation was successful, the shop and the travel agent will keep records of that on their own databases. A sketch of a typical session can be seen as an appendix in [AFG$^+$03].

We have built a prototype of this system using J2EE technology to experiment with the expected functionality and to uncover the basic operations and communications demanded by such e-businesses systems. On the other hand we built different models to experiment with different modelling languages, different tools, and to compare the support they offer to a development team.

5 Modelling Approaches

Our models are in widely used notations that have defined semantics and tool support. These notations are capable to deal with essential notions for e-business applications like concurrency and synchronous/asynchronous message passing. These frameworks allows the creation of simple and abstract models that can be simulated and rigorously checked.

Due to space constraints we cannot offer complete models but, we provide a brief description of them as appendixes A and B to give the reader a flavor of how they look like. The complete models fully documented can be seen in the appendixes given in [AFG$^+$03]. Next we provide a sketch of the basic structures we need and the functionality we expect from each major part of the system.

Communication between the user, the travel agent, and the shops in the prototype is accomplished trough sessions and the underlying web connectivity message system. In our models that was modelled via synchronous channels. We considered (1) a channel to pass requests from the user to the travel agent (2) channels to pass requests from the travel agent to each shop, and (3) channels to get feedback from the shops about if the operation was or was not successful. Another important aspect has to do with the side effects of the interaction in the system. For example, as a result of a successful operation each shop will have to register a change on its database to remember that a resource was taken or released so we need in the models some structures to mimic the databases implemented in the prototype by using JDBC technology. The travel agent has its own database, where all the operations are recorded and its content has to be consistent with all the shops databases except for the intermediate state where a record was made in a shop database but still was not transferred to the travel agent database. But, because the communication is assumed to be synchronous that will eventually occur and because decisions in the system are based only on the shop's database contents this do not cause any harm in the system. Of course, the travel agent will know that if a request has not been answered then the information cannot be considered as an up to date account of the system.

6 Checking Techniques

After running this experiment we were able to collect some interesting experiences. On a higher level we can say that by building the models we were forced to revise and double check the relationships between all the important parts of the system. This also suggests that a realistic expectation is therefore that modelling a system is about four times quicker than prototyping it. While the prototype involved several weeks from a team of three programmers each model was about one and a half week for one person effort. In all cases the people involved had the same level of expertise required to use the necessary technology during both the prototyping and the modelling stages of the development. We do not of course propose developers should construct multiple models. We did so ourselves only to compare notations and tools.

Both tools assisted the modelling stage with syntax and type checking, basic model checking, e.g. infinite loops and deadlock detection, and animation facilities. After no more basic errors were found some simulations were carried out to compare the behavior of the model with the behavior of the prototype and the one expected from the system. By building this models of the system we have been able to check behavioral properties that allowed us to pinpoint some interesting aspects of the system:

Example **1** *(Credit card loop) Part of the user interaction with the system involves providing an authorized credit card brand. The initial prototype allowed users to introduce an unbounded number of attempts to provided their credit card brand. Both tools, SPIN and CIA, allowed us to detect that.*

Example **2** *(Deadlocks) Communications between user, travel agent and shops was implemented via synchronous channels. Sometimes during the construction of the model the interaction of the different processes was very important to detect how interdependent the different parts of the system were to each other. This was especially well considered in SPIN were there is a graphical interface focused on channel communication.*

Example **3** *(detecting subtler errors). An error was introduced in purpose during the construction of the prototype to experiment how we were able to detect it at modelling time. The error is related to the strategy that the Travel Agency has to handle second reservations. This strategy was left unfinished so that when the* *travel agency is asked to book a room in a hotel for a second time by the same user, it tries to book the room in the same hotel used for the first booking. When failing to find another room available the travel agency will not try to book the room in another hotel. Instead it will consider the operation unsuccessful. We were able to detect the potential anomaly during simulation and then confirm it by model checking.*

7 Relating Both Modelling Experiments

Some results emerged from this experience between Promela/SPIN and CSP(LP)/CIA as tools to guide the first stages of modelling:

1. Both demanded almost the same level of knowledge and effort.
2. CSP(LP) is more declarative and hence allows shorter models to be written.
3. Although Promela allows asynchronous channels, CSP(LP) has extra expressiveness due to the logic programming extension (see for example the database implementation provided in appendix B). The concept of queue can be implemented allowing for asynchronous messaging in CSP(LP).
4. SPIN currently offers more support for building the model.
5. Channel handling demands more work in CSP or CSP(LP) specifications which also have the positive side-effect of forcing the user to have a more detailed knowledge about that important side of the system.
6. Trace extraction is currently easier with SPIN.
7. CSP(LP) allows to complement CSP with the use of logic programming features which extends considerably the flexibility of the specification language. An evidence of the importance of this can be seen on [ALBF03] where the flexibility of the input language was a key feature in allowing model checking of a Business Specification Language.

In summary, both tools proved to be very useful in terms of building a simplified version of the system with a slight advantage of SPIN, of being a system developed over more than one decade. In consequence it offers better interface and more information to the system but on the other side there is no impediment for the CSP(LP)/CIA combination to evolve in the same direction.

8 StAC, a More Specific Business Modelling Language

StAC (Structured Activity Compensation) is a language that, in addition to CSP-like operators [Hoa85], offers a set of operators to handle the notion of *compensation.* In StAC it is possible to associate to an action a set of compensation actions providing a way to repair an undesired situation. Compensations are expressed as pairs with the form $P \div Q$, meaning that Q is the compensation planned in case that the effect of P needs to be compensated at a later stage. As the system evolves, compensations are remembered. If all the activities are successfully accomplished then the operator *accept*, ☑, releases the compensations. If any activity fails then the operator *reverse*, ☒, orders the system to apply all the recorded compensations for the current scope. In some contexts the failure to accomplish an activity can be so critical that demands the abortion of a process, that is the role of the *early termination* operator. Both compensation and termination operators can be bounded to a scope of application.

Definition 1 Let A represent an activity, b a boolean condition, P and Q two generic processes, x a variable and X a set of values. Then, we can define as follows the set of well formed formulas in StAC:

$$
\begin{array}{rlllll}
\mathit{Process} ::= & A & \text{(activity label)} & & & \\
| & 0 & \text{(skip)} & | & b \rightarrow P & \text{(condition)} \\
| & \mathit{rec}(P) & \text{(recursion)} & | & P ; Q & \text{(sequence)} \\
| & P || Q & \text{(parallel)} & | & || x \in X . P_x & \text{(generalised parallel)} \\
| & P [\,] Q & \text{(choice)} & | & [\,] x \in X . P_x & \text{(generalised choice)} \\
| & \odot & \text{(early termination)} & | & \{P\} & \text{(termination scoping)} \\
| & P \div Q & \text{(compensation pair)} & | & [P] & \text{(compensation scoping)} \\
| & \boxtimes & \text{(reverse)} & | & \text{☑} & \text{(accept)}
\end{array}
$$

In the example below, processes written in boldface are intended to be basic activities. Each StAC specification is coupled with a B machine [Abr96] describing the state of the system and its basic activities. Basically a B machine is composed by a declaration of sets, variables, invariants, initialisation and operations over those structures. Each StAC activity in a specification will have associated an operation in the corresponding B machine explaining how that activity is implemented in logical terms. We address the reader who wants a more in-detail account of StAC to [CGV$^+$02] and [BF00].

8.1 Travel Agency Example

The travel agency example presented in this section extends the previous travel agency example. In this version the user requests a collection of services instead of a single service, and the travel agency will then try to provide all the requested services. In the StAC model we associate a compensation activity to each service reservation, as the recovery mechanism if any reservation fails or the client decides to cancel his/her requests. A trip is arranged by getting an itinerary, followed by verifying the client's credit card, and depending on whether the card is accept or rejected the reservation is continued or abandoned:

$$
\begin{array}{ll}
\mathit{Trip} = \mathit{GetItinerary} ; \mathbf{VerifyCreditCard} ; & (\mathbf{accepted} \rightarrow \mathit{ContinueReservation} \\
& [\,] \\
& \neg\mathbf{accepted} \rightarrow \mathbf{clearItinerary})
\end{array}
$$

Getting an itinerary involves continually iterating over offering the client the choice of selecting from a car or a hotel "until" ($\star$, defined by using recursion [Fer03]) **EndSelection** is invoked.

$$
\mathit{GetItinerary} = (\mathbf{SelectCar} \; [\,] \; \mathbf{SelectHotel}) \star \mathbf{EndSelection}
$$

ContinueReservations starts by making the reservations on the client's itinerary. If some of the reservations failed, the client is contacted; otherwise, the process ends. The car and hotel reservations are made concurrently.

$$
\begin{aligned}
\mathit{ContinueReservation} = \mathit{MakeReservations}; (&\mathbf{okReservations} \rightarrow \mathit{EndTrip} \\
&\Box \\
&\neg\mathbf{okReservations} \rightarrow \mathit{ContactClient})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{MakeReservations} &= \mathit{CarReservations} \parallel \mathit{HotelReservations} \\
\mathit{CarReservations} &= \parallel_{c \in CAR} . \mathit{CarReservation}(c) \\
\mathit{HotelReservations} &= \parallel_{h \in HOTEL} . \mathit{HotelReservation}(h)
\end{aligned}
$$

The *CarReservation* process reserves a single car using the **Reserve** activity. The travel agency uses two compensation tasks: compensation task S, representing compensation for reservations that have been booked successfully, and compensation task F, representing compensation for reservations that have failed. The choice between which task to add the compensation to is determined by the outcome of the **ReserveCar** activity. Since we use two compensation tasks, instead of having a compensation pair we have a compensation triple, with a primary process P and two compensations Q_1 and Q_2. We model this triple with a construction of the form: $P; (c \rightarrow (\mathit{null} \div_1 Q_1)) \Box (\neg c \rightarrow (\mathit{null} \div_2 Q_2))$ If P makes c true, this is equivalent to $P \div_1 Q_1$ with Q_1 being added to compensation task 1. If P makes c false, this is equivalent to $P \div_2 Q_2$ with Q_2 being added to compensation task 2. With this construction it is possible to organize the compensation information into several compensations tasks, where each one of those tasks can later be reversed or accepted independently. All the cars reservations are made concurrently. The car reservation and its compensations is defined as follows:

$$
\begin{aligned}
\mathit{CarReservation}(f) = \mathbf{ReserveCar}(c);& \\
&((\mathbf{carIsReserved}(c) \rightarrow (\mathit{null} \div_S (\mathbf{CancelCar}(c) \parallel \mathbf{RemoveCar}(c)))) \\
&\Box \\
&(\neg\mathbf{carIsReserved}(c) \rightarrow (\mathit{null} \div_F \mathbf{RemoveCar}(c)))
\end{aligned}
$$

The **RemoveCar** activity removes car c from the client's itinerary, while the **CancelCar** activity cancels the reservation of car c with the car rental. If the activity **ReserveCar** is successful, then to compensate it one has to cancel the reservation with the car rental and also remove that car from the client's itinerary. Otherwise, if the car reservation fails its only necessary to remove the car from the client's itinerary in order to compensate, its not necessary to cancel the car reservation. The hotel reservations are defined similarly and are omitted here. The *ContactClient* process is called if some reservations failed. In this process the client is offered the choice between continuing or quitting:

$$
\begin{aligned}
\mathit{ContactClient} = &(\mathbf{Continue}; \boxtimes_F; \mathit{GetItinerary}; \mathit{ContinueReservations}) \\
&\Box \\
&(\mathbf{Quit}; (\boxtimes_S \parallel \boxtimes_F))
\end{aligned}
$$

In the case that the client decides to continue, reverse is invoked on compensation task F, the failed reservations. This has the effect of removing all failed reservations from the clients itinerary. Compensation task S is preserved as the successful reservations may need to be compensated at a later stage. The client continues by adding more items to the itinerary, which are then reserved. In the case that the client decides to quit, reversal is invoked on both compensation threads. This has the effect of removing all reservations from the clients itinerary and cancelling all successful reservations. Finally, a successful trip reservation is ended by accepting both compensation tasks: $EndTrip = \boxtimes_S \parallel \boxtimes_F$.

8.2 Executable Semantics

One benefit of using StAC is that it would not be possible to capture advanced aspects of the system with Promela (see [ABF03]) and CSP. Modelling with StAC will focus in the higher levels of the system. Any of the previously considered languages can be a good complement by using them to model some of the more low-level features of the system as the communication between processes.

During modelling we have used an animator for StAC processes [LAB+01] based on the CSP(PL) animator described in [Leu01]. At the moment it supports step-by-step animation and backtracking of StAC processes, and it can also detect deadlocks. Animation has helped in the verification of the travel agency, just by comparing the animation execution traces with the expected behavior of the specification several error where found:

1. There is a potential infinite loop if any of the services requested by the client fails. In this case the client can then start choosing a new itinerary that may lead again to some of his/hers requested services to fail.
2. The use of two independent compensation threads for the successful and failed reservations uses a complex notation that is difficult to understand. All this is overcome by the animating the model, because the user can observe the evolution of the compensation threads.
3. The initial StAC model did not have the *EndTrip* process, but the animation showed that without *EndTrip* the compensation information would still be available after the client's logout.

9 Future Work

The XTL model checker allows the user to model check a wide range of system specifications, (see for example [LM99] and [LM02]) the only requirement being that the specification is made by using high level Prolog predicates describing how the system makes transitions between its different states. In this section we describe some basic aspects of XTL and exemplify how to use it to model check StAC specifications. XTL has been implemented using XSB Prolog (http://xsb.sourceforge.net/). Expressiveness and performance indicators are very encouraging for XTL in the sense that it has been able to model check case studies where other tools like SPIN failed and solved problems at similar performance levels. Some domains where XTL was applied successfully are CSP and

B [LAB^{+}01], Petri nets [LM02] and StAC [ALBF03]. The second phase of this research involves model checking both models by contrasting them against behavioral properties expressed in a formal language, LTL (Linear Temporal Logic) for SPIN and CTL (Computational Tree Logic) for XTL. Some properties had been checked by using SPIN and the next step will be to check equivalent or closely related properties in XTL.

The comparison also highlights that part of SPIN success derives from a nice interface which can be even profitable for non-experts in model checking. Some of these services are available in the animators for CSP(LP) and for StAC while the others can be relatively easy added.

10 Conclusions

We conducted an experiment of modelling a prototype by using different languages which have tool support available. We considered Promela/SPIN and CSP(LP)/CIA which share many features in common but also more specific modelling languages like StAC. We left outside the models several details, e.g. all the web based communication was replaced by synchronous channels, the relation sessions/logins is simplified to a userID, the communication between the travel agency was simplified to a request and a response when in reality it is a two steps dialog. The models can be expanded in any of those directions as needed. A quick summary of our experience follows. It is also worth mentioning we have also applied these methodologies to other e-business related case studies: order fulfillment, e-bookstore and mortgage broker.

Benefits of animation/simulation include a) demonstrating flow of information through the system b) exploring interaction between components c) extraction of traces that could be used for generating test cases. In general, however, animations produced by these tools are not of sufficient visual quality to be useful in end users or customer demonstrations. Benefits of model-checking include a) easy discovery of concurrency flaws, e.g., deadlock b) in depth understanding of protocols (process/object interactions) c) discovery of invariants (database consistency constraints) By comparison, benefits of prototyping include a) more realistic user interfaces b) evolution of a class structure that, we believe, would closely approximates that of the actual implementation c) opportunity to gain knowledge of the actual implementation technologies.

Still, our experiments show that rapid modelling is possible (one or two weeks to develop a model, about four times faster than prototyping). Mature notations and tools such as Promela/Spin provide better automated support for modelling, animation, and model-checking. However, the higher level constructs in CSP(LP) allow more faithful modelling of, for example, database tables. Tool support for this notation is sufficiently mature to provide useful insight, but further improvements would be welcomed. Finally, application specific notations such as StAC allow the most rapid modelling of all. Given that long-running transactions are likely to be the basis of future e-business systems, we believe that it is worthwhile further developing such notations and tools to support them.

References

[ABF03] Juan C. Augusto, Michael Butler, and Carla Ferreira. Using spin and step to verify stac specifications. In *Proceedings of PSI'03, 5th International A.P.Ershov Conference on Perspectives of System Informatics (to be published), Novosibirsk (Russia)*, 2003.

[Abr96] J. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University, 1996.

[AFG$^+$03] J. Augusto, Carla Ferreira, Andy Gravell, Michael Leuschel, and Karen M. Y. NG. Exploring different approaches to modelling in enterprise information systems. Technical report, Electronics and Computer Science Department, University of Southampton, 2003. Technical Report, http://www.ecs.soton.ac.uk/∼jca/rm.pdf.

[ALBF03] Juan C. Augusto, Michael Leuschel, Michael Butler, and Carla Ferreira. Using the extensible model checker xtl to verify stac business specifications. In *Pre-proceedings of 3rd Workshop on Automated Verification of Critical Systems (AVoCS 2003), Southampton (UK)*, pages 253–266, 2003.

[Bec00] Kent Beck. *Extreme Programming Explained.* Addison-Wesley, 2000.

[BF00] M. Butler and C. Ferreira. A process compensation language. In *IFM'2000 – Integrated Formal Methods, volume 1945 of LNCS*, pages 61–76. Springer Verlag, 2000.

[CGV$^+$02] M. Chessell, C. Griffin, D. Vines, M. Butler, C. Ferreira, and P. Henderson. Extending the concept of transaction compensation. *IBM Journal of Systems and Development*, 41(4):743–758, 2002.

[Fer03] C. Ferreira. Precise modelling of business processes with compensation. PhD Thesis (submitted), Electronics and Computer Science Department, University of Southampton, 2003.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[Hol97] Gerard Holzmann. The spin model checker. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[LAB$^+$01] M. Leuschel, L. Adhianto, M. Butler, C. Ferreira, and L. Mikhailov. Animation and model checking of csp and b using prolog technology. In *Proceedings of the ACM Sigplan Workshop on Verification and Computational Logic, VCL'2001*, pages 97–109, 2001.

[Leu01] M. Leuschel. Design and implementation of the high-level specification language csp(lp) in prolog. In *Proceedings of PADL'01*, pages 14–28. editor I. V. Ramakrishnan, LNCS 1990, Springer Verlag, 2001.

[LM99] M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In *Proceedings of Logic-Based Program Synthesis and Transformation (LOPSTR'99)*, pages 63–82. editor Annalisa Bossi, Venice, Italy, LNCS 1817, 1999.

[LM02] Michael Leuschel and Thierry Massart. Logic programming and partial deduction for the verification of reactive systems: An experimental evaluation. In *Proceedings of 2nd Workshop on Automated Verification of Critical Systems (AVOCS'02), Birmingham (UK)*, pages 143–150, 2002.

[LR00] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques.* Prentice Hall PTR, 2000.

[Ros99] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1999.

A Fragment of Promela Model

```
    /* channels for communication between processes */
    chan ch_ta = [queue_length] of {bit, byte, bit};
    chan ch_car00 = [queue_length] of {bit, byte};
    ...
    chan ch_car00_2_ta = [queue_length] of {bit, bit};
    ...
    /* databases */
    byte cars00[resources_length];
    byte cars01[resources_length];
    byte rooms10[resources_length];
    byte rooms11[resources_length];
    DBrecord taDB[ta_records];
proctype user() {
i=0;
do /* repeat choices */
:: (i < logins) ->
   if /* choose a user ID in {1..3} */
   :: loginID = 1
   :: loginID = 2
   :: loginID = 3
   fi;
   i++;
   checkCreditCard(ccbit1, ccbit2);
   if
   :: correctCreditCard ->
        if
        :: ch_ta!0, loginID, 0 /* unbook a car  */
        :: ch_ta!1, loginID, 0 /* book a car    */
        :: ch_ta!0, loginID, 1 /* unbook a room */
        :: ch_ta!1, loginID, 1 /* book a room   */
        fi
   :: else -> atomic{ printf("Incorrect credit card !!");
   fi
:: (i >= logins) -> break
od
}
...
proctype ta() {
end:
  do
  :: ch_ta?0, userid,0 -> ch_car00!0,userid; CUnbooking(0, 0)
  :: ch_ta?1, userid,0 -> ch_car00!1,userid; CBooking(0, 0)
  :: ch_ta?0, userid,0 -> ch_car01!0,userid; CUnbooking(0, 1)
  :: ch_ta?1, userid,0 -> ch_car01!1,userid; CBooking(0, 1)
  ... (idem for Hotels)
  od
}
init{run user(); run ta();
     run car00(); run car01(); run hotel10(); run hotel11()
}
```

B Fragment of CSP(LP) Model

```
agent User(integer) : {tadb, h11db};
User(_logins) =
if (_logins > 5) then STOP
   else
   ((CheckCreditCard(1, _logins)) []
    (CheckCreditCard(2, _logins)) []
    (CheckCreditCard(3, _logins)) );
...
agent TA:{ch_ta,ch_car00,ch_car01,ch_hotel10,ch_hotel11};
TA = ch_ta?0?_userID?0 ->
   (((ch_car00!0!_userID -> SKIP) [| {ch_car00} |] CarRental00)
    []
    ((ch_car01!0!_userID -> SKIP) [| {ch_car01} |] CarRental01) );
TA = ch_ta?1?_userID?0 ->
   (((ch_car00!1!_userID -> SKIP) [| {ch_car00} |] CarRental00)
    []
    ((ch_car01!1!_userID -> SKIP) [| {ch_car01} |] CarRental01) );
... (idem for Hotels)

-- Travel Agent database
agent TADB(multiset) : {tadb};
TADB(nil) = tadb!empty -> TADB(nil);
TADB(_State) = tadb?member._x: (_x in _State) -> TADB(_State);
TADB(_State) = tadb?add?_x -> TADB(cons(_x,_State));
TADB(_State) = tadb?rem?_x: _x in _State -> TADB(rem(_State,_x));
TADB(_State) = tadb?nexists?_x: not(_x in _State) -> TADB(_State);

agent MAIN : {};
MAIN =
(TADB(nil) [| {tadb} |]
(C00DB(nil) [| {c00db} |] (C01DB(nil) [| {c01db} |]
(H10DB(nil) [| {h10db} |] (H11DB(nil) [| {h11db} |] User(1) )))));
```