A Refinement Framework for Autonomous Agents

Qin Li and Graeme Smith

School of Information Technology and Electrical Engineering
The University of Queensland, Australia
q.li2@uq.edu.au, smith@itee.uq.edu.au

Abstract. An autonomous agent is one that is not only directed by its environment, but is also driven by internal motivation to achieve certain goals. The popular Belief-Desire-Intention (BDI) design paradigm allows such agents to adapt to environmental changes by calculating a new execution path to their current goal, or when necessary turning to another goal. In this paper we present an approach to modelling autonomous agents using an extension to Object-Z. This extension supports both data and action refinement, and includes the use of LTL formulas to describe an agent's desire as a sequence of prioritised goals. It turns out, however, that the introduction of desire-driven behaviour is not monotonic with respect to refinement. We therefore introduce an additional refinement proof obligation to enable the use of simulation rules when checking refinement.

Keywords: Autonomous agents, BDI agents, Refinement, Object-Z, Temporal logic

1 Introduction

The design of autonomous agents is one of the central issues of the artificial intelligence community [1]. An agent has the capability to manage its own resources and sense its environment. The further behaviour of an agent is often determined dynamically based on its current perception of itself and the environment as well as a goal to achieve. This is the main difference between agents and conventional components.

An autonomous agent is usually described in terms of not only its "physical" features such as variables and actions but also its "mental" features such as *beliefs*, *desires* and *intentions* (BDI) [2, 3]. An autonomous agent in the BDI paradigm formulates a plan (its intention) based on its current beliefs about itself and its environment in order to achieve its desire. Its behaviour, therefore, is derived not only from what it is able to do, but also from what it wants to do [1].

In this paper, we use an extension to Object-Z [4] to specify autonomous agents. The interactions between an agent and its environment are recorded with the inputs and outputs within action definitions. If the information obtained from the environment violates the belief of the agent (which is implicitly included in the precondition of the actions), we consider this to be divergence which can be refined in the development process. This perspective allows us to refine an agent to adapt to a hostile environment by introducing reaction mechanisms for unexpected inputs.

The desire-driven behaviour of autonomous agents is captured by restricting an agent's behaviour to paths leading to its desire. A desire is specified as a sequence of goals each specified in terms of linear temporal logic (LTL) [5]. Typical goals include getting a task done in the future which can be specified with the eventually temporal operator $\Diamond$, or maximising a reward at each step which can be specified with the always temporal operator $\Box$.

To be general, a goal can refer to the interaction variables to reflect the influence of the environment on the motivation of the agent. For example, this allows goals of the form $\Box(env\_state \Rightarrow \Diamond agent\_state)$ where the agent adapts the nature of its goal according to the environment. However, it also allows goals of the form $\Diamond env\_state$ where the environment reaches a desirable state. The latter are unrealistic in the sense that the agent cannot control its environment, and will therefore not influence the behaviour of the agent.

The goals are ordered with priority within the desire and we assume that the agent will have only one goal at any moment. Initially, the goal of the agent will be set to the goal of the desire with the highest priority. The agent follows an execution path which leads to the current goal taking into account interaction with the environment. If there is no path to achieve the goal, the agent sets another goal of the desire to its current goal. If none of the goals of the desire is achievable, the agent acts in an arbitrary fashion, which we call *unmotivated* behaviour.

The refinement theory we provide is able to justify the correctness of design and development paradigms for adapting to environments: (a) introducing local mechanisms to increase the feasibility of the desire under different environmental conditions, and (b) introducing secondary goals to the desire to reduce unmotivated behaviour. To provide flexible support for (a) we allow the introduction of both variables and actions in the concrete specification. Therefore, the refinement theory and its simulation rules are based on event refinement in Event-B [6].

To support (b), we allow unmotivated behaviour to be restricted by further goals. However, as we show, restricting unmotivated behaviour is not monotonic with respect to refinement. In order to refine autonomous agents, therefore, we provide an additional refinement obligation. A refinement of an autonomous agent can be verified by checking both the standard simulation rules and the new proposed obligation.

The remainder of the paper is organised as follows. The specification notation and its semantics are presented in Section 2; the refinement relation and simulation rules are introduced in Section 2.1. Section 3 proposes the specification of an autonomous agent with an explicit desire presented as a sequence of LTL formulas representing goals. In Section 4 the non-monotonicity of the behaviour restriction is revealed, and the refinement obligation to aid the checking of refinement is proposed. Section 5 mentions related work and Section 6 concludes the paper and refers to future research directions.

2 Agents

At low levels of abstraction an agent can be modelled as a state machine, or state-transition system. But an autonomous agent is conveniently specified more abstractly by stating explicitly its desire. This distinguishes an agent from a general reactive component; an agent adjusts its choice of actions to meet its desire [1]. This will be considered in Section 3. For now we represent an agent syntactically by a construct, based on the class construct of Object-Z [4], which we will call a *module*.

A module includes a state schema declaring the local variables and an invariant constraining their values, an initial state schema and a set of actions modelling state transitions. As in Object-Z, primed variables, *e.g.*, u', denote the value of state variables in the post-state of an action, and actions include a Δ-list of variables whose values they may change.

Unlike standard Object-Z, an action has both a guard and a precondition. The guard condition is stated explicitly in an action separated from the effect predicate describing the action's behaviour. The explicit guard is an extension to Object-Z aimed at allowing a more flexible notion of refinement similar to that of Event-B. Specifically, an action can be enabled in a state which is not included as a pre-state of the effect predicate; but the result is divergence.

An action is of the following form where y denotes those state variables not in the Δ-list.

Action	
$\Delta(x)$	variables which action may change
$u? : \mathit{Type\_of\_u}$	input variables
$v! : \mathit{Type\_of\_v}$	output variables
$a : \mathit{Type\_of\_a}$	auxiliary variables
$\mathit{guard}(u?, a, x, y)$	
$\mathit{effect}(u?, a, x, y, x', v!)$	

In the standard semantics of Object-Z, the state variables are hidden (*i.e.*, executions are represented by sequences of actions) and the interaction variables (inputs and outputs) appear as part of the actions which occur. While such a semantics is suited to standard data refinement [7], to allow the introduction of actions we require a semantics in which the actions are hidden (*i.e.*, executions are represented by sequences of states), and hence embed the interaction variables in the states.

The interaction variables are implicitly added to the Δ-list of every action. Any reference in an action to an input variable is a reference to its pre-state value. Hence, actions cannot refer to or constrain their post-state values which represent the values of the inputs used by the next action. Any reference to an output variable is a reference to its post-state value. In the case that an action does not generate a value for a given output variable $v!$ then $v!$ is implicity assigned the special null (undefined) value ϵ.

Semantically, a module is a tuple $M = (\Sigma, I, \mathcal{A})$ where

- Σ is the set of states of the module. Each state is a function mapping the local variables and interaction variables to values which satisfy the variables' types.

  $$\Sigma \;\widehat{=}\; \{\sigma \mid \sigma \in ((\mathit{Var} \cup \mathit{In}) \to \mathit{Val}) \cup (\mathit{Out} \to (\mathit{Val} \cup \{\epsilon\})\}$$

  where Var is the set of local variables declared in the module, In is the set of input variables appearing in any action of the module, and Out is the set of output variables appearing in any action of the module. Val is the set of all values and ϵ is the null value for output variables.
- $I \subseteq \Sigma$ is the set of states which satisfy the module's initial condition. The initial value of each output variable is ϵ.

  $$I \;\widehat{=}\; \{\sigma \mid \sigma \in \Sigma \wedge \sigma \models (\mathit{init}(\mathit{Var}) \wedge \mathit{inv}(\mathit{Var})) \wedge \forall\, v! \in \mathit{Out} \bullet \sigma(v!) = \epsilon\}$$

  where $\mathit{init}(\mathit{Var})$ is the initialisation condition and $\mathit{inv}(\mathit{Var})$ is the invariant over the state variables. The input variables are not constrained initially. The values chosen represent the values of the inputs used by the first action to occur.
- $\mathcal{A} \subseteq \Sigma \times \Sigma^{\perp}$ is the transition relation specified by the actions where $\Sigma^{\perp} = \Sigma \cup \{\perp\}$ and $\perp$ denotes a divergent state in which the values of the state variables are undefined. Divergence occurs when the current state enables an action but the effect of executing the action is undefined. Divergent behaviour is modelled as maximally nondeterministic behaviour allowing it to be refined by any other behaviour. Hence, divergence can be used to abstract the details of behaviour of interest only at some lower level of abstraction. When an action results in divergent behaviour, the divergent state $\perp$, as well as any other state in Σ, may result. In this way, divergent behaviour can be distinguished semantically from maximally nondeterministic terminating behaviour.

  Formally, an individual action named A is represented semantically as

  $$\begin{aligned} \mathit{sem}\ A \;\widehat{=}\; \{(\sigma, \sigma') \mid \sigma \in \Sigma \wedge \sigma' \in \Sigma^{\perp} \wedge \sigma \models A.\mathit{guard} \wedge \\ ((\sigma, \sigma') \models E(A) \vee \nexists \sigma'' \bullet (\sigma, \sigma'') \models E(A))\} \end{aligned}$$

  where $A.\mathit{guard}$ is the guard condition of action A and

  $$E(A) \;\widehat{=}\; A.\mathit{effect} \wedge \mathit{inv}(\mathit{Var}) \wedge \mathit{inv}(\mathit{Var}') \wedge \forall\, v! \in (\mathit{Out} \backslash A.\mathit{out}) \bullet \sigma'(v!) = \epsilon$$

  where $A.\mathit{effect}$ is the effect predicate of action A, and $A.\mathit{out}$ is its set of output variables.

Given that the set of all action names is *Actions*, we have

$$\mathcal{A} = \bigcup_{A \in Actions} sem\ A$$

For simplicity, we omit the notation *sem* in the rest of the paper when it causes no confusion.

The behaviour of a module is the set of all possible traces of the agent, *i.e.*, infinite sequences of states $\langle \sigma_1, \sigma_2, \ldots \rangle$ where every state is a member of $\Sigma^{\perp}$, $\sigma_1 \in I$, and for all $i \in \mathbb{N}_1$, (σ_i, σ_{i+1}) corresponds to the execution of an action A, or to agent inactivity. By allowing unlimited agent inactivity, we model the fact that an unmotivated autonomous agent can always choose to do nothing. This is not the case when the agent is motivated by a desire.

Formally, the set of traces of a module is defined below where for any $i \in \mathbb{N}_1$, $s[i]$ denotes the ith state in trace s.

Definition 1. *(Module Traces) For a trace s and a set $m \subseteq \mathbb{N}_1$, let $non\_div(s, m)$ be true iff s does not diverge at indices in m, i.e.,*

$$non\_div(s, m) \mathrel{\widehat{=}} \forall\, i \in m \bullet (s[i], s[i+1]) \in \mathcal{A} \cup Skip \wedge (s[i], \perp) \notin \mathcal{A}$$

where $Skip \mathrel{\widehat{=}} \{(\sigma, \sigma') \mid \forall\, x \in Var \bullet \sigma'(x) = \sigma(x) \wedge \forall\, v! \in Out \bullet \sigma'(v!) = \epsilon\}$

The behaviour of any module M is modelled as a set of traces, divided into the following subsets distinguished by divergence.

$nml(M)$ denotes the set of all normal, i.e., non-divergent, traces of M.

$$nml(M) \ \mathrel{\widehat{=}}\ \{s \mid s[1] \in I \wedge non\_div(s, \mathbb{N}_1)\}$$

For $n \in \mathbb{N}_1$, $div(M, n)$ denotes the set of traces of M that diverge after the nth state.

$$div(M, n) \ \mathrel{\widehat{=}}\ \{s \mid s[1] \in I \wedge non\_div(s, \{1 \mathrel{..} n-1\}) \wedge (s[n], \perp) \in \mathcal{A}\}$$

Note that it is not possible to recover from divergence; the behaviour following the nth state is undefined and hence maximally nondeterministic (all behaviours including those with the divergent state $\perp$ are included).

$tr(M)$ denotes the set of all possible execution traces of M.

$$tr(M) \ \mathrel{\widehat{=}}\ nml(M) \cup (\bigcup_{n \geq 1} div(M, n))$$

As an example, consider an agent driving a car. The agent receives information about the local traffic via an on-board navigation device. This information includes the time required to reach the destination on the current route, taking into account traffic congestion, and the time on any alternative routes which are faster than the current route. The agent can change the route it takes based on this information.

Let *Route* be a given type denoting the set of all routes and let the agent's state have two variables *current* : *Route* denoting the current route and *time* :

$Route \mathrel{+\!\!\!\rightarrow} \mathbb{N}$ denoting the travel times of the current route and all alternative routes. Initially, the current route is the only route for which a time is displayed. Action *Update* models the agent receiving route information from the navigator, and action *ChooseRoute* models the agent choosing a route based on the latest information.

Agent

$current : Route$
$time : Route \mathrel{+\!\!\!\rightarrow} \mathbb{N}$

INIT

$\operatorname{dom} time = \{current\}$

Update

$\Delta(time)$
$time? : Route \mathrel{+\!\!\!\rightarrow} \mathbb{N}$

$true$

$current \in \operatorname{dom} time?$
$\forall\, r : \operatorname{dom} time? \setminus \{current\} \bullet time?(r) < time?(current)$
$time' = time?$

ChooseRoute

$\Delta(current)$

$true$

$current' \in \operatorname{dom} time$

Let $R1, R2, R3 \in Route$. A normal trace of the agent is:

$$\begin{aligned}\langle &(current = R1, time = \{R1 \mapsto 50\}, time? = \{R1 \mapsto 40, R2 \mapsto 35\}),\\ &(current = R1, time = \{R1 \mapsto 40, R2 \mapsto 35\}, time? = \{R1 \mapsto 35, R3 \mapsto 20\}),\\ &(current = R3, time = \{R1 \mapsto 35, R3 \mapsto 20\}, time? = \{R3 \mapsto 20\}\}, \ldots\rangle\end{aligned}$$

A divergent trace of the agent is:

$$\begin{aligned}\langle &(current = R1, time = \{R1 \mapsto 50\}, time? = \{R1 \mapsto 40, R2 \mapsto 35\}),\\ &(current = R1, time = \{R1 \mapsto 40, R2 \mapsto 35\}, time? = \{R1 \mapsto 35, R2 \mapsto 40\}),\\ &\bot, \ldots\rangle\end{aligned}$$

In this case, the divergence is caused by the navigator providing an input violating the precondition $\forall\, r : \operatorname{dom} time? \setminus \{current\} \bullet time?(r) < time?(current)$.

2.1 Refinement and Simulation Rules

A refinement of an agent specification guarantees that the changes of the state variables in the concrete specification are consistent with those in the abstract specification with respect to a *retrieve relation* R. Given two modules M_1 and M_2, a retrieve relation $R : \mathbb{P}(\Sigma_1^\perp \times \Sigma_2^\perp)$ defines a correspondence between their states. Note that R maps the divergent state $\perp$ only to itself, *i.e.*, $R(\perp) = \{\perp\}$ and $R^{-1}(\perp) = \{\perp\}$. As well as allowing R to be applied as a function on sets of states (note that a single state argument is interpreted as the singleton set containing that state), we allow it to be applied as a function on traces, sets of traces and formulas. The results of the application of R to these constructs are based on its application to states. For instance, the application of R to a trace can be defined as:

$$R(s) = \{t \mid \forall\, i \in \mathbb{N}_1 \bullet t[i] \in R(s[i])\}$$

The application of R to a set of traces S can be defined based on its application to traces:

$$R(S) \mathrel{\widehat{=}} \bigcup_{s \in S} R(s)$$

The application of R to a formula P is also defined in terms of its application to traces:

$$R(P) = \{Q \mid \forall\, s, t \bullet t \in R(s) \Rightarrow (s \models P \Leftrightarrow t \models Q)\}$$

Definition 2. *(Refinement) Let M_1 and M_2 be two modules. We say M_1 is refined by M_2 with respect to retrieve relation R, denoted $M_2 \sqsupseteq_R M_1$, iff $tr(M_2) \subseteq R(tr(M_1))$.*

The subscript R in $\sqsupseteq_R$ may be omitted if R is the identity relation.

Internal changes in the concrete specification may be hidden by the retrieve relation, making some of the concrete actions appear like inactivity at the abstract level. Such concrete actions are called 'stuttering actions'. Formally, action A defined in a concrete module is called a *stuttering action* if it behaves as inactivity in the abstract view, *i.e.*, $R;\ A \subseteq \mathit{Skip};\ R$. Any concrete action not having that property is called a *change action*.

To prove refinement via Definition 2 is generally intractable, requiring analysis of all traces of the modules, and so as usual we consider simulations. The following simulation rules are inspired by those of Event-B [6] which allow a single abstract state transition to be refined by a sequence of concrete transitions.

Theorem 1. *(Forward Simulation) Let M_1 and M_2 be modules and R be a retrieve relation between their states. Then $M_2 \sqsupseteq_R M_1$ if*

(1) $I_2 \subseteq R(I_1)$

(2) for any change action A_2 of M_2, there exists an action A_1 of M_1 where $R\,;A_2 \subseteq A_1\,; R$.

The proof is straightforward noting that any concrete trace which has stuttering states is related to an abstract trace with inactivity, *i.e.*, *Skip*, in the corresponding positions in the trace.

Condition (2) allows the guards of a concrete change action to be stronger than that of the corresponding abstract action. This can result in the introduction of deadlock, *i.e.*, where no actions are enabled. While an agent can choose to be inactive, we would not usually want to refine an agent to one which can only be inactive in certain circumstances. Hence, as in Event-B, we propose an additional condition to prevent the introduction of deadlock.

(3) The overall guard of M_2 is weaker than that of M_1, *i.e.*,

$$R(\{\sigma \mid \sigma \models g(M_1)\}) \ \subseteq \ \{\gamma \mid \gamma \models g(M_2)\}$$

where the overall guard $g(M)$ is the disjunction of all action guards declared in module M.

3 Autonomous agents

A module can specify the behaviour of an agent by referring to the variables it controls and the actions it can perform. In order to specify the autonomous behaviour of an agent, we need to also specify the motivation for its execution: to fulfil its desire. Hence we add a component to the standard module to represent the agent's desire.

An autonomous agent comprising a module M and desire Q will be denoted by $M?Q$. The desire Q is a finite sequence of goals, *i.e.*, $Q = \langle \Phi_1, \Phi_2, ..., \Phi_n \rangle$. Each goal is represented by a linear temporal logic (LTL) [5] formula.

Initially, the agent sets its goal to the first element of the desire sequence. To achieve this goal, the agent calculates execution paths based on its current beliefs about itself and the environment and chooses to follow a path leading to the goal. If there is no path for the agent to achieve its current goal, the agent changes its goal to the next element of the desire sequence. If no goal in the desire sequence is feasible, the agent's behaviour becomes unmotivated (choosing any enabled action). At a lower level of abstraction, the unmotivated behaviour might be refined by, for example, introducing a new goal to the desire sequence.

We define the desire-driven behaviour of an autonomous agent $M?Q$ in an inductive manner. As the base case, an agent with an empty desire sequence behaves as the definition of its module, *i.e.*, $M?\langle\rangle \mathrel{\widehat{=}} M$.

Consider an agent with only one goal in its desire sequence, *i.e.*, $M?\langle \Phi \rangle$. The traces of $M?\langle \Phi \rangle$ include:

- Any non-divergent trace of M which satisfies Φ.
- Any divergent trace of M which has satisfied Φ before divergence. After the desire has been satisfied, the agent may act in any manner available to it.
- Any trace of M in which, from a certain point, the agent has no opportunity to make a decision which will lead to its desire being satisfied while before that point the agent made acceptable choices all along the trace. Such a trace corresponds to unmotivated behaviour.

Note that a decision made by the agent is acceptable when there exists a path afterward which can satisfy the desire given a cooperative environment. That is, we do not insist the decisions of the agent guarantee that every path afterward can satisfy the desire. This reflects the fact that such decisions would be based on the agent's beliefs about the future behaviour of the environment which may, or may not, turn out to be true.

To formalise our notion of desire-driven behaviour, we first introduce some notation.

1. We use the notation Δ to denote the set of all sequences of states which do not include $\perp$. For a trace $s \in \Delta$ and LTL formula Φ, we say $s \vDash \Phi$ if and only if the temporal property Φ is satisfied by s.
2. For traces s and t, let $s =_n t$ be true iff s and t share the same prefix of length n, *i.e.*,

   $$s =_n t \mathrel{\widehat=} \forall\, i \in 1..n \bullet s[i] = t[i]$$

3. For a set S of traces satisfying a desire, we let $\Gamma(s, S, i)$ denote a predicate that is true when trace s is not in the set S due to either (a) the value of inputs at point i, or (b) when i is 1, the trace's initial state. These situations are ones in which the agent has no opportunity to make a decision which will lead to the desire being satisfied.
   For case (a), (1) there does not exist a trace in S which shares the prefix of trace s up to point i, and (2) there exists a trace $u \in S$ which shares the prefix of s up to point $i - 1$ and differs from s at point i by the input values only. This case indicates that the desire is unable to be satisfied due to inputs from the environment.
   For case (b), there does not exist a trace in S which shares the same initial state as s. This case indicates that the desire is unable to be satisfied due to the initialisation.
   Let $V \lhd \sigma$ denotes the state σ with variables in set V removed.

   $$\begin{aligned}\Gamma(s, S, i) \mathrel{\widehat=}\ & i = 1 \Rightarrow \nexists r \in S \bullet \mathit{In} \lhd r[1] = \mathit{In} \lhd s[1] \wedge \\ & i > 1 \Rightarrow \nexists r \in S \bullet r =_i s \wedge \\ & \qquad\qquad \exists\, u \in S \bullet u =_{i-1} s \wedge \mathit{In} \lhd u[i] = \mathit{In} \lhd s[i]\end{aligned}$$

   Such a trace s contains unmotivated behaviour after point i. It would be kept in the autonomous behaviour for further refinement.

Definition 3. *(Autonomous behaviour) The behaviour of an autonomous agent $M?\langle\Phi\rangle$ is modeled as a set of traces, divided into the following subsets distinguished by divergence.*

The normal traces of $M?\langle\Phi\rangle$ are those normal traces of M which either satisfy Φ (represented by nmlsucc), or fail to satisfy Φ due to input values or their initial state (represented by nmlfail).

$\mathit{nml}(M?\langle\Phi\rangle) \ \mathrel{\widehat=}\ \mathit{nmlsucc}(M?\langle\Phi\rangle) \cup \mathit{nmlfail}(M?\langle\Phi\rangle)$ *where*

$\mathit{nmlsucc}(M?\langle\Phi\rangle) \ \mathrel{\widehat=}\ \{s \mid s \in \mathit{nml}(M) \wedge s \models \Phi\}$

$nmlfail(M?\langle\Phi\rangle) \;\widehat{=}\; \{s \mid s \in nml(M) \land \exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, nmlsucc(M?\langle\Phi\rangle), i)\}$

The divergent traces of $M?\langle\Phi\rangle$ are those divergent traces which satisfy Φ (i.e., all non-diverging traces that do not differ before the point of divergence satisfy Φ) or fail to satisfy Φ due to input values (represented by divfail).

$div(M?\langle\Phi\rangle, n) \;\widehat{=}\; divsucc(M?\langle\Phi\rangle, n) \cup divfail(M?\langle\Phi\rangle, n)$ *where*

$divsucc(M?\langle\Phi\rangle, n) \;\widehat{=}\; \{s \mid s \in div(M, n) \land \forall\, t \in \Delta \bullet s =_n t \Rightarrow t \models \Phi\}$

$divfail(M?\langle\Phi\rangle, n) \;\widehat{=}\; \{s \mid s \in div(M, n) \land \exists\, i \in 1..n \bullet \Gamma(s, nmlsucc(M?\langle\Phi\rangle), i)\}$

The set of all possible traces of $M?\langle\Phi\rangle$ is

$$tr(M?\langle\Phi\rangle) \;\widehat{=}\; nml(M?\langle\Phi\rangle) \cup (\textstyle\bigcup_{n\geq 1} div(M?\langle\Phi\rangle, n))\,.$$

Reconsider the example agent of Section 2. Let the desire of the agent be that the time cost of the current route is always no greater than the previous current route. This can be specified in LTL as follows.

$$\Box(\exists\, t : \mathbb{N} \bullet t = time(current) \land \bigcirc(time(current) \leq t))$$

In this case, the following normal trace (where the agent does not change to a faster route) would no longer be allowed.

$\langle(current = R1, time = \{R1 \mapsto 40\}, time? = \{R1 \mapsto 45, R2 \mapsto 35\}),$
$(current = R1, time = \{R1 \mapsto 45, R2 \mapsto 35\}, time? = \{R1 \mapsto 35, R3 \mapsto 20\}),$
$\ldots\rangle$

If there is no route provided by the navigator that takes less time than the current route, the agent would have no choice but to violate its current goal. The behaviour after this point is considered to be unmotivated behaviour. Such a case is shown below where the input *time?* in the first state gives the agent no choice to satisfy its goal. The rest of the trace is unmotivated.

$\langle(current = R1, time = \{R1 \mapsto 40\}, time? = \{R1 \mapsto 50, R2 \mapsto 45\}),$
$(current = R2, time = \{R1 \mapsto 50, R2 \mapsto 45\}, time? = \{R2 \mapsto 45, R3 \mapsto 40\}),$
$\ldots\rangle$

For an agent with more than one goal in its desire sequence, the definition is as follows.

Definition 4. *(Introducing goals) Let $M?Q$ be an autonomous agent with $Q \neq \langle\rangle$ and Φ be an LTL property. The introduction of goal Φ will take effect when the agent cannot fulfil its original desire Q. In other words, it will further restrict the unmotivated behaviour of the agent $M?Q$. The behaviour of the autonomous agent $M?(Q \frown \langle\Phi\rangle)$ can be defined as follows.*

– *The nmlsucc behaviour is extended by the traces which share the same prefix as a trace satisfying the original desire Q before point i and satisfying Φ instead from the point $i+1$ where Q cannot be fulfilled.*

$$nmlsucc(M?(Q \frown \langle\Phi\rangle)) \,\widehat{=}\, nmlsucc(M?Q) \cup \{s \mid s \in nmlfail(M?Q) \land \\ \exists\, i : \mathbb{N}_1 \bullet \Gamma(s, nmlsucc(M?Q), i) \land s_{i+1} \models \Phi\}$$

where s_i denotes the postfix of s beginning with the ith state.

– *The nmlfail behaviour contains the traces which cannot satisfy the new desire $Q \frown \langle \Phi \rangle$ due to the inputs or their initial state.*

$$\begin{aligned} \mathit{nmlfail}(M?(Q \frown \langle \Phi \rangle)) \mathrel{\widehat{=}} \ & \{s \mid s \in \mathit{nml}(M?Q) \wedge \\ & \exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, \mathit{nmlsucc}(M?(Q \frown \langle \Phi \rangle)), i)\} \end{aligned}$$

– *A similar definition is made for the divergent behaviours.*

$$\begin{aligned} \mathit{divsucc}(M?(Q \frown \langle \Phi \rangle), n) \mathrel{\widehat{=}} \ & \mathit{divsucc}(M?Q, n) \cup \\ & \{s \mid s \in \mathit{divfail}(M?Q, n) \wedge \forall\, t \in \Delta \bullet s =_n t \wedge \\ & \exists\, i \in 1..n \bullet \Gamma(t, \mathit{nmlsucc}(M?Q), i) \wedge t_{i+1} \models \Phi\} \end{aligned}$$

$$\begin{aligned} \mathit{divfail}(M?(Q \frown \langle \Phi \rangle), n) \mathrel{\widehat{=}} \ & \{s \mid s \in \mathit{div}(M?Q, n) \wedge \\ & \exists\, i \in 1..n \bullet \Gamma(s, \mathit{nmlsucc}(M?Q \frown \langle \Phi \rangle), i)\} \end{aligned}$$

According to the above definition, the newly introduced goal only takes effect when the original goals are infeasible. This further restricts the behaviour of the autonomous agent by reducing its unmotivated behaviours. It is intuitive to obtain the conclusion that introducing a goal to the agent's desire sequence refines its behaviour.

Theorem 2. *Let M be a module and Q_1 and Q_2 be desires such that Q_1 is a proper subsequence of Q_2. Then we have $M?Q_2 \sqsupseteq M?Q_1$.*

Proof

1. *If $Q_1 = \langle\rangle$ and $Q_2 = \langle \Phi \rangle$, then from Definition 3, all traces of $M?\langle \Phi \rangle$ are traces of M, i.e., $tr(M?\langle \Phi \rangle) \subseteq tr(M)$. Hence, since $M?\langle\rangle \mathrel{\widehat{=}} M$, we have $tr(M?Q_2) \subseteq tr(M?Q_1)$ and therefore $M?Q_2 \sqsupseteq M?Q_1$.*
2. *If $Q_1 \neq \langle\rangle$ and $Q_2 = Q_1 \frown \langle \Phi_1, ..., \Phi_n \rangle$, then $tr(M?Q_2)$ can be obtained recursively from Definition 4 one goal at a time. It is straightforword to show that in each recursive step, any trace which does not satisfy the original goals nor the newly introduced goal but at some point has the opportunity to achieve the new goal is removed from the behaviour of the previous step. Hence we have $tr(M?Q_2) \subseteq tr(M?Q_1)$ and therefore $M?Q_2 \sqsupseteq M?Q_1$.*

4 Refinement Obligation

The desire-driven behaviour of an autonomous agent only removes a trace where it can choose another one with a shared prefix to achieve the desire. Otherwise, the behaviour is like a standard module without desires. This allows us to refine an agent by introducing additional goals to reduce the unmotivated behaviour of the abstract specification. However, such a restriction is non-monotonic with respect to the refinement order. An intuitive example is shown below.

Consider the situation shown in Figure 1 which shows a subset of the behaviours of two agents M_1 and M_2. s is the only trace which shares a prefix with t of length k and satisfies a goal Φ.

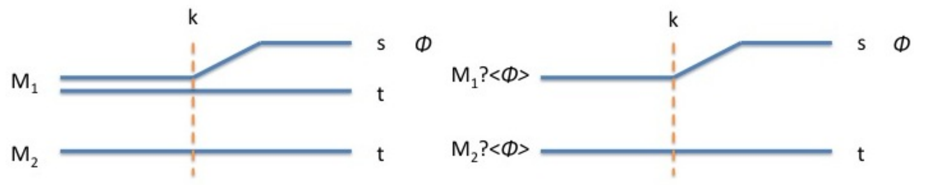

Fig. 1. Counter example for monotonicity

Consider the case where the traces s and t of agent M_1 differ at point $k+1$ due to a local choice made by the agent. At point k, therefore, the agent has a chance to make a local choice and follow trace s to fulfil the goal Φ. So trace t will be removed from the behaviour of $M_1?\langle\Phi\rangle$.

Agent M_2 has all traces of M_1 apart from s. It is obvious that $M_2 \sqsupseteq M_1$ according to Definition 2. However, with the same desire $\langle\Phi\rangle$, trace t will not be removed from the behaviour of $M_2?\langle\Phi\rangle$ since, in this case, the agent has no opportunity to fulfil the desire. Hence the desire-driven behaviour $M_1?\langle\Phi\rangle$ is not refined by $M_2?\langle\Phi\rangle$.

This situation arises whenever we disable a choice which can lead to an agent's desire and hence make it impossible for the refined agent to satisfy the desire. To avoid such refinements, we need an additional proof obligation that ensures that the concrete agent does not introduce more unmotivated behaviour. That is, if a concrete trace is a trace with unmotivated behaviour from a given point, then its corresponding abstract trace is also a trace with unmotivated behaviour from the same point.

This conclusion can be formalised by the following theorem. For simplicity, we first explore the case of introducing a goal to the empty desire sequence.

Theorem 3. *(Refinement Obligation)*

Let M_1 and M_2 be modules linked by a retrieve relation R and $\Phi_2 \in R(\Phi_1)$. We have $M_2?\langle\Phi_2\rangle \sqsupseteq_R M_1?\langle\Phi_1\rangle$ if both of the following conditions hold.

1. *$M_2 \sqsupseteq_R M_1$*
2. *For any trace of M_2 if it is impossible for the agent to make a local choice leading to the satisfaction of Φ_2 at a point in the trace, then from the same point in the corresponding trace of M_1 it is also impossible to satisfy Φ_1. That is, for all $s \in tr(M_1)$ and $t \in tr(M_2)$ where $t \in R(s)$, we have*

$$\forall\, i \in \mathbb{N}_1 \bullet (\Gamma(t, nmlsucc(M_2?\langle\Phi_2\rangle), i) \Rightarrow \Gamma(s, nmlsucc(M_1?\langle\Phi_1\rangle), i))$$

Proof:

Following Definition 2, we need to show that for any trace $t \in tr(M_2?\langle\Phi_2\rangle)$, *each of its corresponding traces s, i.e., those traces where $t \in R(s)$, satisfy $s \in tr(M_1?\langle\Phi_1\rangle)$. According to condition 1 and Theorem 2, we get $s \in tr(M_1)$ (since $M_2?\langle\Phi_2\rangle \sqsupseteq M_2 \sqsupseteq_R M_1$).*

The proof proceeds by a case analysis of traces s and t based on whether or not they are in Δ (the set of traces without $\perp$) and, if so, on their satisfaction of the respective desires. Since $s \in R(t)$ and $R(\perp) = \{\perp\} \wedge R^{-1}(\perp) = \{\perp\}$, it follows that $t \in \Delta \Leftrightarrow s \in \Delta$.

Assume $s, t \in \Delta$. In this situation, there are two cases to consider:

1. *If $t \models \Phi_2$ then its corresponding trace s satisfies Φ_1 since $\Phi_2 \in R(\Phi_1)$ (see the definition of R applied to formulas). Hence, s appears in $tr(M_1?\langle\Phi_1\rangle)$.*
2. *If $t \not\models \Phi_2$ then, according to Definition 3, its appearance in $tr(M_2?\langle\Phi_2\rangle)$ implies that there exists a point in t where the agent has no opportunity to make a choice leading to the goal Φ_2.*

   $$\exists\, i \in \mathbb{N}_1 \bullet \Gamma(t, nmlsucc(M_2?\langle\Phi_2\rangle), i)$$

   Hence in the corresponding trace s, Γ holds at the same point i due to condition 2. This then leads to

   $$\exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, nmlsucc(M_1?\langle\Phi_1\rangle), i)$$

   Hence, according to Definition 3, s also appears in $tr(M_1?\langle\Phi_1\rangle)$.

The remaining case to consider is when $s, t \notin \Delta$. According to Definition 1, we can find $n > 1$ so that t diverges at the nth place, i.e., *$t \in div(M_2?\langle\Phi_2\rangle, n)$.*

1. *If $t \in divsucc(M_2?\langle\Phi_2\rangle, n)$, which means $\forall\, u \in \Delta \bullet t =_n u \Rightarrow u \models \Phi_2$, then since $\Phi_2 \in R(\Phi_1)$ and $t \in R(s)$, we have $\forall\, r \in \Delta \bullet s =_n r \Rightarrow r \models \Phi_1$. Hence $s \in div(M_1?\langle\Phi_1\rangle, n)$.*
2. *If $t \in divfail(M_2?\langle\Phi_2\rangle, n)$, which means $\exists\, i \in 1..n \bullet \Gamma(t, nmlsucc(M_2?\langle\Phi_2\rangle), i)$, then according to condition 2 we can conclude that $\exists\, i \in 1..n \bullet \Gamma(s, nmlsucc(M_1?\langle\Phi_1\rangle), i)$. Hence we have $s \in div(M_1?\langle\Phi_1\rangle, n)$.*

In summary, we have $t \in tr(M_2?\langle\Phi_2\rangle) \Rightarrow s \in tr(M_1?\langle\Phi_1\rangle)$ which means $M_2?\langle\Phi_2\rangle \sqsupseteq_R M_1?\langle\Phi_1\rangle$.

The refinement obligation implies that the refined agent has no more unmotivated behaviour than the abstract agent. Reconsider the counter example of Figure 1. The trace s is a trace with unmotivated behaviour in $M_2?\langle\Phi\rangle$ but not in $M_1?\langle\Phi\rangle$.

A general version of the refinement obligation for all possible desire sequences is as follows.

Theorem 4. *(General Obligation) Let $M_1?Q_1$ and $M_2?Q_2$ be two autonomous agents linked by a retrieve relation R, and Φ_1 and Φ_2 be LTL formulas with $\Phi_2 \in R(\Phi_1)$. We have $M_2?(Q_2 \frown \langle\Phi_2\rangle) \sqsupseteq_R M_1?(Q_1 \frown \langle\Phi_1\rangle)$ if both of the following conditions hold.*

1. $M_2?Q_2 \sqsupseteq_R M_1?Q_1$
2. *For all* $s \in tr(M_1?Q_1)$ *and* $t \in tr(M_2?Q_2)$ *where* $t \in R(s)$ *we have*

$$\begin{aligned} &\forall\, i \in \mathbb{N}_1 \bullet \mathit{\Gamma}(t, nmlsucc(M_2?Q_2 \frown \langle \mathit{\Phi}_2 \rangle), i) \\ &\qquad \Rightarrow \mathit{\Gamma}(s, nmlsucc(M_1?Q_1 \frown \langle \mathit{\Phi}_1 \rangle), i) \end{aligned}$$

The proof is similar to that of Theorem 3.

With the refinement obligation we obtain from Theorems 3 and 4, refinement checking between two autonomous agents can be done monotonically and recursively. First we can check the refinement between their modules without desires. Then we check the obligation for each goal along their desire sequences.

The refinement obligation requires the refined agent preserve the possibility of achieving the desire under any environmental inputs. A development strategy which sufficiently satisfies the obligation is to introduce local mechanisms to adapt to environmental “hostility”. The strategy includes the following three rules.

1. Weakening the precondition of actions to accept a larger range of inputs. This makes the agent handle more situations of the environment and hence reduces divergence.
2. Optimizing the decision making algorithms to improve the local decision so that the agent has a more deterministic way to achieve the desire under certain inputs than the specification does.
   For example, for the desire to take a faster route, the operation *ChooseRoute* of Section 2 could be refined to always choose the fastest route.

   ChooseRoute
   $\mathit{\Delta}(current)$

   $true$

   $current' \in \operatorname{dom} time$
   $time(current') = min(\operatorname{ran} time)$

3. Introducing additional goals to regulate the unmotivated behaviours.
   For example, for the unmotivated behaviour where the navigator provides no faster route than the current one, we introduce a secondary goal to allow the agent to choose the fastest of the provided routes.

$$\begin{aligned} &\square((\exists\, r : \operatorname{dom} time \bullet time(r) < time(current) \Rightarrow \\ &\qquad \bigcirc(\nexists r : \operatorname{dom} time \bullet time(r) < time(current)) \end{aligned}$$

   With this secondary goal, the following trace from Section 3 is no longer an unmotivated behaviour.

   $\langle (current = R1, time = \{R1 \mapsto 40\}, time? = \{R1 \mapsto 50, R2 \mapsto 45\}),$
   $(current = R2, time = \{R1 \mapsto 50, R2 \mapsto 45\}, time? = \{R2 \mapsto 45, R3 \mapsto 40\}),$
   $\ldots \rangle$

Rule 1 is a conventional refinement rule. Rule 2 requires the designer to ensure the agent has a way to achieve its desire in the development. Rule 3 allows the designer to design a sequence of complementary goals when the agent fails to satisfy its current goal.

Following the above strategy, the refined agent has more reliable local mechanisms to achieve the desire than it does in the abstract specification.

5 Related work

There is some formal work developed to specify interactions between an agent and its environment. Alternating transition systems (ATS) proposed by Alur *et al.* [8] treats an agent and its environment as the opponents of a game. Agents choose their own transitions to update the current state and the final result is the intersection of their choices. The action-based alternating transition system studied by Atkinson *et al.* [9] provides reasoning techniques for which action should be chosen by an agent in particular situations.

Zhu [10] proposes a formal notation for specifying agent behaviour. The autonomous behaviour of the agent is formalised by a set of rules designed for various environmental scenarios.

While the above approaches are able to specify agent-environment interactions, they do not specify the behaviour of agents as being driven by its 'mental' states (*e.g.*, desire).

Rao *et al.* [2] use a possible world model to interpret the semantics of BDI logic for autonomous agents. This is suitable for representing the belief, desire and intention of agents by assigning each of them a set of accessible worlds. However, unlike our approach, it lacks a theory to justify the correctness of the development of autonomous agents by introducing mechanisms to adapt to the environment.

Aştefănoaei and de Boer [11] define a notion of refinement for BDI agents. Unlike our approach abstract and concrete specifications are not in the same notation. Therefore, their approach allows only a single refinement step from an abstract to a concrete representation of an agent, not the incremental development of an agent. More importantly, in their approach the goal of an agent is fixed. While they allow environmental hostility to be dealt with by changing plans, they don't allow the goal of an agent to be changed, nor the possibility of an agent not fulfilling its goal.

6 Conclusion

In this paper we provided a formal refinement framework to justify the correctness of the development of autonomous agents. Agents are specified in an extension of Object-Z including a desire specified by a sequence of LTL properties. The autonomous behaviour of an agent is realised by restricting its ordinary behaviour with the goals in its desire. Although this behaviour restriction is not monotonic with respect to refinement, we proposed an additional refinement obligation to allow checking refinement using ordinary simulation rules. The refinement framework can support the development of an autonomous agent by either introducing local mechanisms to adapt to environmental updates or introducing secondary goals to reduce unmotivated behaviour.

As a first step, this paper does not illustrate the development strategy in detail. We intend to define it formally in future work. We will also consider adding explicit agent beliefs to our framework which we expect will be refined by introducing preconditions into concrete actions. The ultimate goal of this work is to examine how the analysis of individual autonomous agents can be used in the analysis of multi-agent systems.

Acknowledgement This work was supported by Australian Research Council (ARC) Discovery Grant DP110101211.

References

1. Wooldridge, M.: An Introduction to MultiAgent Systems (2nd Edition). Wiley (2009)
2. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: 1st International Conference of Multi-agent Systems (ICMAS-95), MIT Press (1995) 312–319
3. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review **10** (1995) 115–152
4. Smith, G.: The Object-Z Specification Language. Kluwer Academic Publishers (2000)
5. Emerson, E.: Temporal and modal logic. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science. Volume B. Elsevier (1990) 996–1072
6. Abrial, J.R.: Modelling in Event-B. Cambridge University Press (2010)
7. Derrick, J., Boiten, E.: Refinement in Z and Object-Z, Foundations and Advanced Applications. Springer-Verlag (2001)
8. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: 9th International Conference on Concurrency Theory (CONCUR '98). Volume 1466 of LNCS., Springer-Verlag (1998) 163–178
9. Atkinson, K., Bench-Capon, T.: Practical reasoning as presumptive argumentation using action based alternating transition systems. Artif. Intell. **171**(10-15) (2007) 855–874
10. Zhu, H.: Formal specification of agent behaviour through environment scenarios. In Rash, J.L., Truszkowski, W., Hinchey, M.G., Rouff, C.A., Gordon, D., eds.: Formal Approaches to Agent-Based Systems. Volume 1871 of Lecture Notes in Computer Science., Springer-Verlag (2001) 263–277
11. Aştefănoaei, L., de Boer, F.: The refinement of multi-agent systems. In Dastani, M., Hindriks, K., Meyer, J.J., eds.: Specification and Verification of Multi-agent Systems. Springer-Verlag (2010) 35–65